Capitolo 7

Stallo dei processi

OBIETTIVI

- **Descrizione approfondita delle situazioni di stallo (*deadlock*) che impediscono il completamento del lavoro a gruppi di processi concorrenti.**
- **Presentazione dei metodi differenti per prevenire o impedire le situazioni di stallo.**

In un ambiente con multiprogrammazione più processi possono competere per ottenere un numero finito di risorse; se una risorsa non è correntemente disponibile, il processo richiedente passa allo stato d'attesa. Se le risorse richieste sono trattenute da altri processi, a loro volta nello stato d'attesa, il processo potrebbe non cambiare più il suo stato. Situazioni di questo tipo sono chiamate di **stallo** (*deadlock*). Questo argomento è trattato brevemente anche nel Capitolo 6, nello studio dei semafori.

Un efficace esempio di situazione di stallo si può ricavare da una legge dello stato del Kansas approvata all'inizio del XX secolo, che in una sua parte recita: "Quando due treni convergono a un incrocio, ambedue devono arrestarsi, e nessuno dei due può ripartire prima che l'altro si sia allontanato".

In questo capitolo si descrivono i metodi che un sistema operativo può usare per prevenire o affrontare le situazioni di stallo. Occorre in ogni caso notare che la maggior parte dei sistemi operativi attuali non offre strumenti di prevenzione di queste situazioni. Funzioni adatte a questo scopo saranno probabilmente aggiunte presto. Date le tendenze correnti, il problema dello stallo può diventare soltanto più importante: aumento del numero dei processi e delle risorse all'interno di un sistema, programmi multithread e preferenza attribuita ai sistemi con archivi di file e basi di dati anziché ai sistemi a lotti.

7.1 Modello del sistema

Un sistema è composto da un numero finito di risorse da distribuire tra più processi in competizione. Le risorse sono suddivise in tipi differenti, ciascuno formato da un certo numero di istanze identiche. Cicli di CPU, spazio di memoria, file e dispositivi di I/O (come stampanti e lettori DVD), sono tutti esempi di tipi di risorsa. Se un sistema ha due unità d'elaborazione, tale tipo di risorsa ha due istanze. Analogamente, il tipo di risorsa *stampante* può avere cinque istanze.

Se un processo richiede un'istanza relativa a un tipo di risorsa, l'assegnazione di *qualsiasi* istanza di quel tipo può soddisfare la richiesta. Se ciò non si verifica significa che le istanze non sono identiche e le classi di risorse non sono state definite correttamente. Un sistema può, per esempio, avere due stampanti; se a nessuno interessa sapere quale sia la stampante in funzione, le due stampanti si possono definire come appartenenti alla stessa classe di risorse; se, però, una stampante si trova al nono piano e l'altra al piano terra, allora le due stampanti si possono considerare non equivalenti, e per definire ciascuna delle due può essere necessario ricorrere a classi di risorse distinte.

Prima di adoperare una risorsa, un sistema deve richiederla e, dopo averla usata, deve rilasciarla. Un processo può richiedere tutte le risorse necessarie per eseguire il compito assegnatogli, anche se ovviamente il numero delle risorse richieste non può superare quello totale delle risorse disponibili nel sistema: un processo non può richiedere tre stampanti se il sistema ne ha solo due.

Nelle ordinarie condizioni di funzionamento un processo può servirsi di una risorsa soltanto se rispetta la seguente sequenza.

1. **Richiesta.** Se la richiesta non si può soddisfare immediatamente – per esempio, perché la risorsa è attualmente in possesso di un altro processo – il processo richiedente deve attendere finché non possa acquisire tale risorsa.
2. **Uso.** Il processo può operare sulla risorsa (se, per esempio, la risorsa è una stampante, il processo può effettuare una stampa).
3. **Rilascio.** Il processo rilascia la risorsa.

La richiesta e il rilascio di risorse avvengono tramite chiamate di sistema, come illustrato nel Capitolo 2. Ne sono esempi le chiamate di sistema `request()` e `release()`, `open()` e `close()`, `allocate()` e `free()`. La richiesta e il rilascio di altre risorse si possono eseguire per mezzo delle operazioni `wait()` e `signal()` su semafori. Quindi, ogni volta che si usa una risorsa, il sistema operativo controlla che il processo utente ne abbia fatto richiesta e che questa gli sia stata assegnata. Una tabella di sistema registra lo stato di ogni risorsa e, se questa è assegnata, indica il processo relativo. Se un processo richiede una risorsa già assegnata a un altro processo, il processo richiedente può essere accodato agli altri processi che attendono tale risorsa.

Un gruppo di processi entra in stallo quando tutti i processi del gruppo attendono un evento che può essere causato solo da un altro processo che si trova nello stato di attesa. Gli eventi maggiormente discussi in questo capitolo sono l'acquisizione e il rilascio di risorse. Le risorse possono essere sia fisiche, per esempio, stampanti, unità a nastri, spazio di memoria e cicli di CPU, sia logiche, per esempio, file, semafori e monitor. Tuttavia anche altri eventi possono condurre a situazioni di stallo, come le funzioni di IPC trattate nel Capitolo 3.

Per esaminare una situazione di stallo si consideri un sistema con tre lettori CD. Si supponga che tre processi ne possiedano uno ciascuno. Se ogni processo richiedesse un altro lettore CD, i tre processi entrerebbero in stallo: ciascuno attenderebbe il rilascio del lettore, ma quest'evento potrebbe essere causato solo da uno degli altri processi che a loro volta attendono l'ulteriore lettore CD. Quest'esempio descrive uno stallo causato da processi che competono per acquisire lo stesso tipo di risorsa.

Le situazioni di stallo possono implicare anche diversi tipi di risorsa. Si consideri, per esempio, un sistema con una stampante e un lettore DVD, e si supponga che il processo P_i sia in possesso del lettore e il processo P_j della stampante. Se P_i richiedesse la stampante e P_j il lettore DVD, si avrebbe uno stallo.

Un programmatore che sviluppa applicazioni multithread deve prestare una particolare attenzione a questo problema: i programmi multithread sono ottimi candidati alle situazioni di stallo, poiché più thread possono competere per le risorse condivise.

7.2 Caratterizzazione delle situazioni di stallo

In una situazione di stallo, i processi non terminano mai l'esecuzione, e le risorse del sistema vengono bloccate impedendo l'esecuzione di altri processi. Prima di trattarne le soluzioni, descriviamo le caratteristiche del problema dello stallo.

7.2.1 Condizioni necessarie

Si può avere una situazione di stallo *solo se* si verificano contemporaneamente le seguenti quattro condizioni.

1. **Mutua esclusione.** Almeno una risorsa deve essere non condivisibile, vale a dire che è utilizzabile da un solo processo alla volta. Se un altro processo richiede tale risorsa, si deve ritardare il processo richiedente fino al rilascio della risorsa.
2. **Possesso e attesa.** Un processo in possesso di almeno una risorsa attende di acquisire risorse già in possesso di altri processi.
3. **Impossibilità di prelazione.** Non esiste un diritto di prelazione sulle risorse, vale a dire che una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, dopo aver terminato il proprio compito.
4. **Attesa circolare.** Deve esistere un insieme $\{P_0, P_1, ..., P_n\}$ di processi, tale che P_0 attende una risorsa posseduta da P_1, P_1 attende una risorsa posseduta da P_2, ..., P_{n-1} attende una risorsa posseduta da P_n e P_n attende una risorsa posseduta da P_0.

STALLO CON LOCK MUTEX

Vediamo come si possa incorrere in situazioni di stallo in un programma multithread di Pthread che usi i lock mutex. La funzione `pthread_mutex_init()` inizializza un semaforo mutex su cui non è attivo un lock. I lock mutex sono acquisiti e restituiti per mezzo di `pthread_mutex_lock()` e `pthread_mutex_unlock()`, rispettivamente. Se un thread tenta di acquisire un mutex già impegnato, l'invocazione di `pthread_mutex_lock()` blocca il thread finché il possessore del mutex non invochi `pthread_mutex_unlock()`.

Il brano si codice seguente genera due lock mutex:

```
/* Crea e inizializza due lock mutex */
pthread_mutex_t primo_mutex;
pthread_mutex_t secondo_mutex;

pthread_mutex_init(&primo_mutex, NULL);
pthread_mutex_init(&secondo_mutex, NULL);
```

Si creano poi due thread, di nome `thread_uno` e `thread_due`, che possono accedere a entrambi i lock mutex. Sono eseguiti dalle funzioni `esegui_uno()` e `esegui_due()`, rispettivamente, come mostrato nella Figura 7.1.

In questo esempio, `thread_uno` tenta di acquisire i lock mutex nell'ordine (1) `primo_mutex`, (2) `secondo_mutex`, mentre `thread_due` tenta di acquisire i lock nell'ordine (1) `secondo_mutex`, (2) `primo_mutex`. Lo stallo è possibile nell'ipotesi che `thread_uno` acquisisca `primo_mutex`, mentre `thread_due` acquisisca `secondo_mutex`.

Si noti che lo stallo, pur essendo possibile, non si verifica se `thread_uno` è in grado di acquisire e rilasciare entrambi i lock prima che `thread_due` tenti a sua volta di impossessarsene. L'esempio evidenzia un problema importante per la gestione dello stallo: è difficile identificare e sottoporre a test gli stalli che si verificano solo in determinate condizioni.

```
/* thread_uno esegue in questa funzione */
void *esegui_uno(void *param)
{
   pthread_mutex_lock(&primo_mutex);
   pthread_mutex_lock(&secondo_mutex);
   /*
    * Fa qualcosa
    */
   pthread_mutex_unlock(&secondo_mutex);
   pthread_mutex_unlock(&primo_mutex);

   pthread_exit(0);
}

/* thread_due esegue in questa funzione */
void *esegui_due(void *param)
{
   pthread_mutex_lock(&secondo_mutex);
   pthread_mutex_lock(&primo_mutex);
   /*
    * Fa qualcosa
    */
   pthread_mutex_unlock(&primo_mutex);
   pthread_mutex_unlock(&secondo_mutex);

   pthread_exit(0);
}
```

Figura 7.1 Esempio di stallo.

Occorre sottolineare che tutte e quattro le condizioni devono essere vere, altrimenti non si può avere alcuno stallo. La condizione dell'attesa circolare implica la condizione di possesso e attesa, quindi le quattro condizioni non sono completamente indipendenti; tuttavia è utile considerare separatamente ciascuna condizione (Paragrafo 7.4).

7.2.2 Grafo di assegnazione delle risorse

Le situazioni di stallo si possono descrivere con maggior precisione avvalendosi di una rappresentazione detta **grafo di assegnazione delle risorse**. Si tratta di un insieme di vertici V e un insieme di archi E, con l'insieme di vertici V composto da due sottoinsiemi: $P = \{P_1, P_2, ..., P_n\}$, che rappresenta tutti i processi del sistema, e $R = \{R_1, R_2, ..., R_m\}$, che rappresenta tutti i tipi di risorsa del sistema.

Un arco diretto dal processo P_i al tipo di risorsa R_j si indica $P_i \rightarrow R_j$, e significa che il processo P_i ha richiesto un'istanza del tipo di risorsa R_j, e attualmente attende tale risorsa. Un arco diretto dal tipo di risorsa R_j al processo P_i si indica $R_j \rightarrow P_i$, e significa che un'istanza del tipo di risorsa R_j è assegnata al processo P_i. Un arco orientato $P_i \rightarrow R_j$ si chiama **arco di richiesta**, un arco orientato $R_j \rightarrow P_i$ si chiama **arco di assegnazione**.

Graficamente ogni processo P_i si rappresenta con un cerchio e ogni tipo di risorsa R_j si rappresenta con un rettangolo. Giacché il tipo di risorsa R_j può avere più di un'istanza, ciascuna di loro si rappresenta con un puntino all'interno del rettangolo. Occorre notare che un arco di richiesta è diretto soltanto verso il rettangolo R_j, mentre un arco di assegnazione deve designare anche uno dei puntini del rettangolo.

Quando il processo P_i richiede un'istanza del tipo di risorsa R_j, si inserisce un arco di richiesta nel grafo di assegnazione delle risorse. Se questa richiesta può essere esaudita, si trasforma *immediatamente* l'arco di richiesta in un arco di assegnazione, che al rilascio della risorsa viene cancellato.

Nel grafo di assegnazione delle risorse della Figura 7.2 è illustrata la seguente situazione.

- Insiemi P, R ed E:
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$
- Istanze delle risorse:
  - un'istanza del tipo di risorsa R_1
  - due istanze del tipo di risorsa R_2

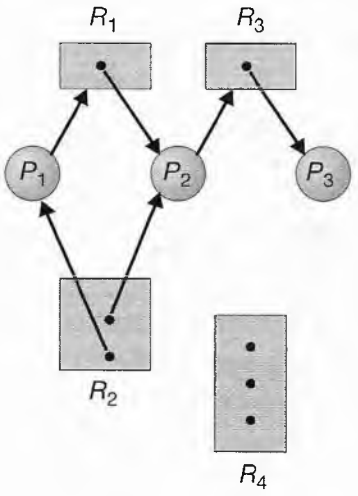

Figura 7.2 Grafo di assegnazione delle risorse.

- ◇ un'istanza del tipo di risorsa R_3
- ◇ tre istanze del tipo di risorsa R_4

◆ Stati dei processi:

- ◇ il processo P_1 possiede un'istanza del tipo di risorsa R_2 e attende un'istanza del tipo di risorsa R_1
- ◇ il processo P_2 possiede un'istanza dei tipi di risorsa R_1 ed R_2 e attende un'istanza del tipo di risorsa R_3
- ◇ il processo P_3 possiede un'istanza del tipo di risorsa R_3

Data la definizione di grafo di assegnazione delle risorse, è facile mostrare che, se il grafo non contiene cicli, nessun processo del sistema subisce uno stallo; se il grafo contiene un ciclo, può sopraggiungere uno stallo.

Se ciascun tipo di risorsa ha esattamente un'istanza, allora l'esistenza di un ciclo implica la presenza di uno stallo; se il ciclo riguarda solo un insieme di tipi di risorsa, ciascuno dei quali ha solo un'istanza, si è verificato uno stallo. Ogni processo che si trovi nel ciclo è in stallo. In questo caso l'esistenza di un ciclo nel grafo è una condizione necessaria e sufficiente per l'esistenza di uno stallo.

Se ogni tipo di risorsa ha più istanze, un ciclo non implica necessariamente uno stallo. In questo caso l'esistenza di un ciclo nel grafo è una condizione necessaria ma non sufficiente per l'esistenza di uno stallo.

Per spiegare questo concetto conviene ritornare al grafo di assegnazione delle risorse della Figura 7.2. Si supponga che il processo P_3 richieda un'istanza del tipo di risorsa R_2. Poiché attualmente non è disponibile alcuna istanza di risorsa, si aggiunge un arco di richiesta $P_3 \rightarrow R_2$ al grafo, com'è illustrato nella Figura 7.3. A questo punto nel sistema ci sono due cicli minimi:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

I processi P_1, P_2 e P_3 sono in stallo: il processo P_2 attende la risorsa R_3, posseduta dal processo P_3; il processo P_3, invece, attende che il processo P_1 o P_2 rilasci la risorsa R_2; inoltre il processo P_1 attende che il processo P_2 rilasci la risorsa R_1.

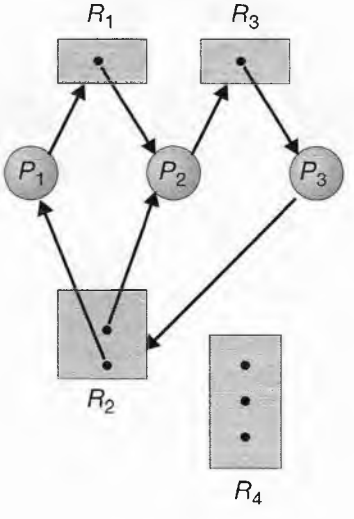

Figura 7.3 Grafo di assegnazione delle risorse con uno stallo.

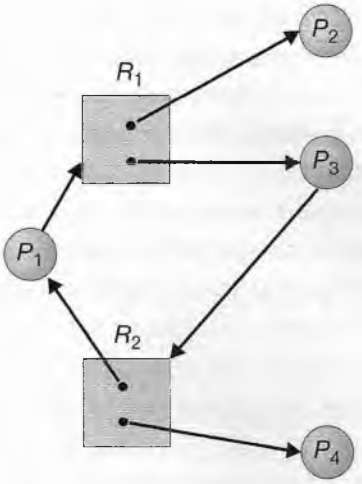

Figura 7.4 Grafo di assegnazione delle risorse con un ciclo, ma senza stallo.

Si consideri ora il grafo di assegnazione delle risorse della Figura 7.4. Anche in questo esempio c'è un ciclo:

$$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$

In questo caso, però, non si ha alcuno stallo: il processo P_4 può rilasciare la propria istanza del tipo di risorsa R_2, che si può assegnare al processo P_3, rompendo il ciclo.

Per concludere, l'assenza di cicli nel grafo di assegnazione delle risorse implica l'*assenza* di situazioni di stallo nel sistema. Viceversa, la presenza di un ciclo non è sufficiente a implicare la presenza di uno stallo nel sistema. Questa osservazione è importante ai fini della gestione del problema delle situazioni di stallo.

7.3 Metodi per la gestione delle situazioni di stallo

Essenzialmente, il problema delle situazioni di stallo si può affrontare impiegando tre metodi:

- si può usare un protocollo per prevenire o evitare le situazioni di stallo, assicurando che il sistema non entri *mai* in stallo;
- si può permettere al sistema di entrare in stallo, individuarlo, e quindi eseguire il ripristino;
- si può ignorare del tutto il problema, *fingendo* che le situazioni di stallo non possano mai verificarsi nel sistema.

Quest'ultima è la soluzione usata dalla maggior parte dei sistemi operativi, compresi UNIX e Windows. Nel seguito sono spiegati brevemente tutti questi metodi. Gli algoritmi relativi sono presentati in modo dettagliato nei Paragrafi dal 7.4 al 7.7.

Tuttavia, prima di procedere, vogliamo considerare anche l'opinione di quegli sviluppatori che hanno contestato il fatto che nessuno degli approcci di base si adatti da solo all'intero spettro di problemi di allocazione delle risorse nei sistemi operativi. Tali approcci possono essere combinati, in modo da permettere la selezione del migliore per ciascuna classe di risorse del sistema.

Per assicurare che non si verifichi mai uno stallo, il sistema può servirsi di metodi di prevenzione o di metodi per evitare tale situazione. **Prevenire le situazioni di stallo** significa far uso di metodi atti ad assicurare che non si verifichi almeno una delle condizioni necessarie (Paragrafo 7.2.1). Questi metodi, discussi nel Paragrafo 7.4, prevengono le situazioni di stallo prescrivendo il modo in cui si devono fare le richieste delle risorse.

Per **evitare le situazioni di stallo** occorre che il sistema operativo abbia in anticipo informazioni aggiuntive riguardanti le risorse che un processo richiederà e userà durante le sue attività. Con queste informazioni aggiuntive si può decidere se una richiesta di risorse da parte di un processo si può soddisfare o si debba invece sospendere. In tale processo di decisione il sistema tiene conto delle risorse correntemente disponibili, di quelle correntemente assegnate a ciascun processo, e delle future richieste e futuri rilasci di ciascun processo. Questi metodi sono discussi nel Paragrafo 7.5.

Se un sistema non impiega né un algoritmo per prevenire né un algoritmo per evitare gli stalli, tali situazioni possono verificarsi. In un ambiente di questo tipo il sistema può servirsi di un algoritmo che ne esamini lo stato, al fine di stabilire se si è verificato uno stallo e in tal caso ricorrere a un secondo algoritmo per il ripristino del sistema. Tali argomenti sono discussi nei Paragrafi 7.6 e 7.7.

Se un sistema non garantisce che le situazioni di stallo non possano mai verificarsi e non fornisce alcun meccanismo per la loro individuazione e per il ripristino del sistema, situazioni di stallo possono avvenire senza che ci sia la possibilità di capire cos'è successo. In questo caso la presenza di situazioni di stallo non rilevate può causare un degrado delle prestazioni del sistema; infatti la presenza di risorse assegnate a processi che non si possono eseguire determina lo stallo di un numero crescente di processi che richiedono tali risorse, fino al blocco totale del sistema che dovrà essere riavviato manualmente.

Sebbene questo modo di affrontare il problema delle situazioni di stallo non sembri fattibile, è in ogni caso usato in alcuni sistemi operativi perché è più economico dei metodi che altrimenti si dovrebbero adoperare per prevenire le situazioni di stallo, evitarle, o anche individuarle per il successivo ripristino. In molti sistemi, infatti, le situazioni di stallo accadono abbastanza raramente (magari una volta l'anno). In alcune circostanze il sistema è *congelato*, ma non è in stallo. Si consideri, per esempio, la situazione determinata da un processo d'elaborazione in tempo reale che si esegue con la priorità più elevata (o qualsiasi processo in esecuzione in un sistema con scheduler senza diritto di prelazione) e che non restituisce il controllo al sistema operativo. Il sistema deve così disporre di meccanismi non automatici di ripristino per le situazioni che non sono in modo specifico di stallo, che può impiegare anche per le situazioni di stallo.

7.4 Prevenire le situazioni di stallo

Com'è evidenziato nel Paragrafo 7.2.1, affinché si abbia uno stallo si devono verificare quattro condizioni necessarie; perciò si può *prevenire* il verificarsi di uno stallo assicurando che almeno una di queste condizioni non possa capitare. Questo metodo è descritto nei particolari, con una trattazione di ciascuna delle quattro condizioni necessarie.

7.4.1 Mutua esclusione

Deve valere la condizione di mutua esclusione per le risorse non condivisibili: una stampante non può essere condivisa da più processi. Le risorse condivisibili, invece, non richiedono l'accesso mutuamente esclusivo, perciò non possono essere coinvolte in uno stallo. I file

aperti per la sola lettura sono un buon esempio di risorsa condivisibile; è ovviamente consentito l'accesso contemporaneo da parte di più processi. Un processo non deve mai attendere una risorsa condivisibile. Ma poiché alcune risorse sono intrinsecamente non condivisibili, non si possono prevenire in generale le situazioni di stallo negando la condizione di mutua esclusione.

7.4.2 Possesso e attesa

Per assicurare che la condizione di possesso e attesa non si presenti mai nel sistema, occorre garantire che un processo che richiede una risorsa non ne possegga altre. Si può usare un protocollo che ponga la condizione che ogni processo, prima di iniziare la propria esecuzione, richieda tutte le risorse che gli servono e che esse gli siano assegnate. Questa condizione si può realizzare imponendo che le chiamate di sistema che richiedono risorse per un processo precedano tutte le altre.

Un protocollo alternativo è quello che permette a un processo di richiedere risorse solo se non ne possiede: un processo può richiedere risorse e adoperarle, ma prima di richiederne altre deve rilasciare tutte quelle che possiede.

Per spiegare la differenza tra questi due protocolli si può considerare un processo che copi i dati da un DVD a un file in un disco, ordina il contenuto del file e quindi stampa i risultati. Nel caso del primo protocollo, il processo deve richiedere fin dall'inizio il lettore DVD, il file nel disco e una stampante. La stampante rimane in suo possesso per tutta la durata dell'esecuzione, anche se si usa solo alla fine.

Il secondo protocollo prevede che il processo richieda inizialmente solo il lettore DVD e il file nel disco. Il processo copia i dati dal DVD al disco, quindi rilascia le due risorse. A questo punto richiede il file nel disco e la stampante, e dopo aver copiato il file nella stampante rilascia le due risorse e termina.

Entrambi i protocolli presentano due svantaggi principali. Innanzitutto, l'**utilizzo delle risorse** può risultare poco efficiente, poiché molte risorse possono essere assegnate, ma non utilizzate, per un lungo periodo di tempo. Nel caso del secondo metodo dell'esempio precedente si possono rilasciare il lettore DVD e il file nel disco, per poi richiedere il file e la stampante, solo se si ha la certezza che i dati nel file restino immutati tra il rilascio e la seconda richiesta. Se non si può garantire quest'ultima condizione, è necessario richiedere tutte le risorse all'inizio per entrambi i protocolli.

Il secondo svantaggio è dovuto al fatto che si possono verificare situazioni di attesa indefinita. Un processo che richieda più risorse molto usate può trovarsi nella condizione di attenderne indefinitamente la disponibilità, poiché almeno una delle risorse di cui necessita è sempre assegnata a qualche altro processo.

7.4.3 Impossibilità di prelazione

La terza condizione necessaria prevede che non sia possibile avere la prelazione su risorse già assegnate. Per assicurare che questa condizione non persista, si può impiegare il seguente protocollo. Se un processo che possiede una o più risorse ne richiede un'altra che non gli si può assegnare immediatamente (cioè il processo deve attendere), allora si esercita la prelazione su tutte le risorse attualmente in suo possesso. Si ha cioè il rilascio implicito di queste risorse, che si aggiungono alla lista delle risorse che il processo sta attendendo; il processo viene nuovamente avviato solo quando può ottenere sia le vecchie risorse sia quella che sta richiedendo.

In alternativa, quando un processo richiede alcune risorse, si verifica la disponibilità di queste ultime: se sono disponibili vengono assegnate, se non lo sono, si verifica se sono assegnate a un processo che attende altre risorse. In tal caso si sottraggono le risorse desidera-

te a quest'ultimo processo e si assegnano al processo richiedente. Se le risorse non sono disponibili né sono possedute da un processo in attesa, il processo richiedente deve attendere. Durante l'attesa si può avere la prelazione su alcune sue risorse; ciò può accadere solo se un altro processo le richiede. Un processo si può avviare nuovamente solo quando riceve le risorse che sta richiedendo e recupera tutte quelle a esso sottratte durante l'attesa.

Questo protocollo è adatto a risorse il cui stato si può salvare e recuperare facilmente in un secondo tempo, come i registri della CPU e lo spazio di memoria, mentre non si può in generale applicare a risorse come le stampanti e le unità a nastri.

7.4.4 Attesa circolare

La quarta e ultima condizione necessaria per una situazione di stallo è l'attesa circolare. Un modo per assicurare che tale condizione d'attesa non si verifichi consiste nell'imporre un ordinamento totale all'insieme di tutti i tipi di risorse e un ordine crescente di numerazione per le risorse richieste da ciascun processo.

Si supponga che $R = \{R_1, R_2, ..., R_m\}$ sia l'insieme dei tipi di risorse. A ogni tipo di risorsa si assegna un numero intero unico che permetta di confrontare due risorse e stabilirne la relazione di precedenza nell'ordinamento. Formalmente, si definisce una funzione iniettiva, $f: R \rightarrow N$, dove N è l'insieme dei numeri naturali. Se, ad esempio, l'insieme dei tipi di risorsa R contiene unità a nastri, unità a dischi e stampanti, la funzione f si può definire come segue:

$f(\textit{unità a nastri}) = 1$
$f(\textit{unità a dischi}) = 5$
$f(\textit{stampante}) = 12$

Per prevenire il verificarsi di situazioni di stallo si può considerare il seguente protocollo: ogni processo può richiedere risorse solo seguendo un ordine crescente di numerazione. Ciò significa che un processo può richiedere inizialmente qualsiasi numero di istanze di un tipo di risorsa, ad esempio R_i, dopo di che il processo può richiedere istanze del tipo di risorsa R_j se e solo se $f(R_j) > f(R_i)$. Se sono necessarie più istanze dello stesso tipo di risorsa si deve presentare una singola richiesta per tutte le istanze. Ad esempio, con la funzione definita precedentemente, un processo che deve impiegare contemporaneamente l'unità a nastri e la stampante deve prima richiedere l'unità a nastri e poi la stampante. In alternativa, si può stabilire che un processo, prima di richiedere un'istanza del tipo di risorsa R_j, rilasci qualsiasi risorsa R_i tale che $f(R_i) \geq f(R_j)$.

Se si usa uno di questi due protocolli, la condizione di attesa circolare non può sussistere. Ciò si può dimostrare supponendo, per assurdo, che esista un'attesa circolare. Si supponga che l'insieme di processi coinvolti nell'attesa circolare sia $\{P_0, P_1, ..., P_n\}$, dove P_i attende una risorsa R_i, posseduta dal processo P_{i+1}. (Sugli indici si usa l'aritmetica modulare, quindi P_n attende una risorsa R_n posseduta da P_0.) Poiché il processo P_{i+1} possiede la risorsa R_i mentre richiede la risorsa R_{i+1}, è necessario che sia verificata la condizione $f(R_i) < f(R_{i+1})$ per tutti gli i, ma ciò implica che $f(R_0) < f(R_1) < ... < f(R_n) < f(R_0)$. Per la proprietà transitiva, risulta che $f(R_0) < f(R_0)$, il che è impossibile; quindi, non può esservi attesa circolare.

Questo schema si può implementare in un programma applicativo imponendo un ordine a tutti gli oggetti del sistema che richiedano sincronizzazione. Tutte le richieste inerenti a tali oggetti dovranno seguire l'ordine crescente. Per esempio, se l'ordinamento dei lock nel programma della Figura 7.1 fosse stato

$f(\texttt{primo\_mutex}) = 1$
$f(\texttt{secondo\_mutex}) = 5$

allora `thread_due` non avrebbe potuto richiedere i lock nell'ordine inverso. Si tenga presente che la semplice esistenza di un ordinamento delle risorse non protegge dallo stallo: è infatti responsabilità degli sviluppatori di applicazioni stendere programmi che rispettino tale ordinamento. Si noti anche che la funzione f andrebbe definita secondo l'ordine d'uso normale delle risorse del sistema. Per esempio, poiché generalmente l'unità a nastri si usa prima della stampante, è ragionevole definire $f(\textit{unità a nastri}) < f(\textit{stampante})$.

Come si è detto, è responsabilità dei programmatori rispettare l'ordinamento delle risorse; tuttavia, è possibile sviluppare del software in grado di verificare che l'acquisizione dei lock avvenga nell'ordine corretto, ed emetta avvisi appropriati in caso contrario. Uno di tali verificatori, disponibile per le versioni BSD di UNIX (per esempio, FreeBSD), è noto con il nome di **witness**. Esso usa lock mutex per proteggere le sezioni critiche, come descritto nel Capitolo 6, e tiene traccia dinamicamente delle relazioni d'ordine fra i lock del sistema. Si consideri nuovamente il programma della Figura 7.1. Se `thread_uno` è il primo thread ad acquisire i lock, e se ciò avviene nell'ordine (1) `primo_mutex`, (2) `secondo_mutex`, il programma witness registra il fatto che `primo_mutex` deve essere acquisito prima di `secondo_mutex`. Se, in seguito, `thread_due` acquisisce i lock violando quest'ordine, il programma witness produce sulla console del sistema un messaggio di notifica.

7.5 Evitare le situazioni di stallo

Gli algoritmi di prevenzione delle situazioni di stallo trattati nel Paragrafo 7.4 si basano sul controllo delle modalità di richiesta, così da assicurare che non si possa verificare almeno una delle condizioni necessarie perché si abbia uno stallo. Questo metodo può però causare effetti collaterali negativi, come uno scarso utilizzo dei dispositivi e una ridotta produttività del sistema.

Un metodo alternativo per evitare le situazioni di stallo consiste nel richiedere ulteriori informazioni sulle modalità di richiesta delle risorse. In un sistema con un'unità a nastri e una stampante, per esempio, il processo P può dichiarare che intende richiedere prima l'unità a nastri, poi la stampante, e che rilascerà entrambe solo in seguito. Il processo Q, invece, richiederà prima la stampante e poi l'unità a nastri. Una volta acquisita la sequenza completa delle richieste e dei rilasci di ogni processo, si può stabilire per ogni richiesta se il processo debba attendere o meno, per evitare una possibile situazione di stallo futura. In seguito a ogni richiesta, il sistema deve esaminare le risorse attualmente disponibili, le risorse attualmente assegnate a ogni processo e le richieste e i rilasci futuri per ciascun processo.

Gli algoritmi differiscono tra loro per la quantità e il tipo di informazioni richieste. Il modello più semplice e più utile richiede che ciascun processo dichiari il *numero massimo* delle risorse di ciascun tipo di cui necessita. Data un'informazione a priori per ogni processo sul massimo numero di risorse richiedibili per ciascun tipo, si può costruire un algoritmo capace di assicurare che il sistema non entri in stallo. Questo algoritmo definisce un metodo per **evitare lo stallo**, ed esamina dinamicamente lo stato di assegnazione delle risorse per garantire che non possa esistere una condizione di attesa circolare. Lo *stato* di assegnazione delle risorse è definito dal numero di risorse disponibili e assegnate e dalle richieste massime dei processi. Nei due paragrafi successivi esaminiamo due algoritmi di impedimento alle situazioni di stallo.

7.5.1 Stato sicuro

Uno stato si dice *sicuro* se il sistema è in grado di assegnare risorse a ciascun processo (fino al suo massimo) in un certo ordine e impedire il verificarsi di uno stallo. Più formalmente, un sistema si trova in stato sicuro solo se esiste una **sequenza sicura**. Una sequenza di processi $<P_1, P_2, ..., P_n>$ è una sequenza sicura per lo stato di assegnazione attuale se, per ogni P_i, le richieste che P_i può ancora fare si possono soddisfare impiegando le risorse attualmente disponibili più le risorse possedute da tutti i P_j con $j < i$. In questa situazione, se le risorse necessarie al processo P_i non sono disponibili immediatamente, allora P_i può attendere che tutti i P_j abbiano finito, e a quel punto P_i può ottenere tutte le risorse di cui ha bisogno, completare il compito assegnato, restituire le risorse assegnate e terminare. Quando P_i termina, P_{i+1} può ottenere le risorse richieste, e così via. Se non esiste una sequenza di questo tipo, lo stato del sistema si dice *non sicuro*.

Uno stato sicuro non è di stallo. Viceversa, uno stato di stallo è uno stato non sicuro; comunque non tutti gli stati non sicuri sono stati di stallo (Figura 7.5). Uno stato non sicuro *potrebbe* condurre a uno stallo. Finché lo stato rimane sicuro, il sistema operativo può evitare il verificarsi di stati non sicuri e di stallo. In uno stato non sicuro il sistema operativo non può impedire ai processi di richiedere risorse in modo da causare uno stallo: ciò che accade negli stati non sicuri dipende dal comportamento dei processi.

Per illustrare meglio quel che si è detto sopra, si consideri un sistema con 12 unità a nastri magnetici e 3 processi: P_0, P_1 e P_2. Il processo P_0 può richiedere 10 unità a nastri, il processo P_1 può richiederne 4 e il processo P_2 può richiedere fino a 9. Supponendo che all'istante t_0 il processo P_0 possieda 5 unità a nastri, e che i processi P_1 e P_2 ne possiedano 2 ciascuno, restano libere 3 unità a nastri.

	Richieste massime	*Unità possedute*
P_0	10	5
P_1	4	2
P_2	9	2

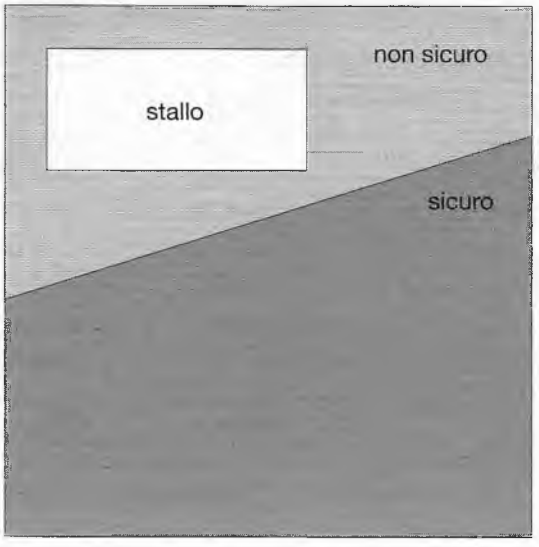

Figura 7.5 Spazi degli stati sicuri, non sicuri e di stallo.

All'istante t_0, il sistema si trova in uno stato sicuro. La sequenza $<P_1, P_0, P_2>$ soddisfa la condizione di sicurezza, poiché al processo P_1 si possono assegnare immediatamente tutte le unità a nastri richieste, che saranno poi restituite (a questo punto sono disponibili 5 unità a nastri), quindi il processo P_0 può avere tutte le unità a nastri richieste e restituirle (il sistema ha 10 unità a nastri disponibili) e infine il processo P_2 potrebbe avere tutte le sue unità a nastri e restituirle (sono disponibili tutte e 12 le unità a nastri).

Un sistema può passare da uno stato sicuro a uno stato non sicuro. Si supponga che all'istante t_1 il processo P_2 richieda un'ulteriore unità a nastri e che questa gli sia assegnata: il sistema non si trova più nello stato sicuro. A questo punto, si possono assegnare tutte le unità a nastri richieste soltanto al processo P_1. Al momento della restituzione, il sistema avrà solo 4 unità a nastri disponibili. Poiché al processo P_0 sono assegnate 5 unità a nastri, ma il numero massimo è 10, il processo può richiederne altre 5, ma poiché queste non sono disponibili il processo P_0 deve attendere. Analogamente, il processo P_2 può richiedere altre 6 unità a nastri ed è costretto ad attendere; il risultato è una situazione di stallo. L'errore è stato commesso nel soddisfare la richiesta di un'ulteriore unità a nastri fatta dal processo P_2. Se P_2 avesse atteso il termine di uno degli altri processi e il conseguente rilascio delle sue risorse, la situazione di stallo si sarebbe potuta evitare.

Dato il concetto di stato sicuro, si possono definire algoritmi che permettano di evitare le situazioni di stallo. L'idea è semplice: è sufficiente assicurare che il sistema rimanga sempre in uno stato sicuro. Il sistema si trova inizialmente in uno stato sicuro; ogni volta che un processo richiede una risorsa, il sistema deve stabilire se la risorsa è attualmente disponibile oppure se il processo debba attendere. Si soddisfa la richiesta solo se l'assegnazione lascia il sistema in uno stato sicuro.

In questo modo, se un processo richiede una risorsa attualmente disponibile può essere comunque costretto ad attendere. Quindi, l'utilizzo delle risorse può essere inferiore rispetto a quello che si avrebbe in assenza di un algoritmo per evitare le situazioni di stallo.

7.5.2 Algoritmo con grafo di assegnazione delle risorse

Quando il sistema per l'assegnazione delle risorse è tale che ogni tipo di risorsa ha una sola istanza, per evitare le situazioni di stallo si può far uso di una variante del grafo di assegnazione delle risorse definito nel Paragrafo 7.2.2. Oltre agli archi di richiesta e di assegnazione, si introduce un nuovo tipo di arco, l'arco di reclamo (*claim edge*). Un **arco di reclamo** $P_i \rightarrow R_j$ indica che il processo P_i può richiedere la risorsa R_j in un qualsiasi momento futuro. Quest'arco ha la stessa direzione dell'arco di richiesta, ma si rappresenta con una linea tratteggiata. Quando il processo P_i richiede la risorsa R_j, l'arco di reclamo $P_i \rightarrow R_j$ diventa un arco di richiesta. Analogamente, quando P_i rilascia la risorsa R_j, l'arco di assegnazione $R_j \rightarrow P_i$ diventa un arco di reclamo $P_i \rightarrow R_j$. Occorre sottolineare che le risorse devono essere reclamate a priori nel sistema. Ciò significa che prima che il processo P_i inizi l'esecuzione, tutti i suoi archi di reclamo devono essere già inseriti nel grafo di assegnazione delle risorse. Questa condizione si può indebolire permettendo l'aggiunta di un arco di reclamo $P_i \rightarrow R_j$ al grafo solo se tutti gli archi associati al processo P_i sono archi di reclamo.

Si supponga che il processo P_i richieda la risorsa R_j. La richiesta si può soddisfare solo se la conversione dell'arco di richiesta $P_i \rightarrow R_j$ nell'arco di assegnazione $R_j \rightarrow P_i$ non causa la formazione di un ciclo nel grafo di assegnazione delle risorse. Occorre ricordare che la sicurezza si controlla con un algoritmo di rilevamento dei cicli, e che un algoritmo per il rilevamento di un ciclo in questo grafo richiede un numero di operazioni dell'ordine di n^2, dove con n si indica il numero dei processi del sistema.

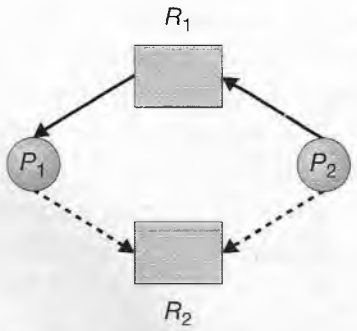

Figura 7.6 Grafo di assegnazione delle risorse per evitare le situazioni di stallo.

Se non esiste alcun ciclo, l'assegnazione della risorsa lascia il sistema in uno stato sicuro. Se invece si trova un ciclo, l'assegnazione conduce il sistema in uno stato non sicuro, e il processo P_i deve attendere che si soddisfino le sue richieste.

Per illustrare questo algoritmo si consideri il grafo di assegnazione delle risorse della Figura 7.6. Si supponga che P_2 richieda R_2. Sebbene sia attualmente libera, R_2 non può essere assegnata a P_2, poiché, com'è evidenziato nella Figura 7.7, quest'operazione creerebbe un ciclo nel grafo e un ciclo indica che il sistema è in uno stato non sicuro. Se, a questo punto, P_1 richiedesse R_2, si avrebbe uno stallo.

7.5.3 Algoritmo del banchiere

L'algoritmo con grafo di assegnazione delle risorse non si può applicare ai sistemi di assegnazione delle risorse con più istanze di ciascun tipo di risorsa. L'algoritmo per evitare le situazioni di stallo qui descritto, pur essendo meno efficiente dello schema con grafo di assegnazione delle risorse, si può applicare a tali sistemi, ed è noto col nome di **algoritmo del banchiere**. Questo nome è stato scelto perché l'algoritmo si potrebbe impiegare in un sistema bancario per assicurare che la banca non assegni mai tutto il denaro disponibile, poiché, se ciò avvenisse, non potrebbe più soddisfare le richieste di tutti i suoi clienti.

Quando si presenta, un nuovo processo deve dichiarare il numero massimo delle istanze di ciascun tipo di risorsa di cui necessita. Questo numero non può superare il numero totale delle risorse del sistema. Quando un utente richiede un gruppo di risorse, si deve stabilire se l'assegnazione di queste risorse lasci il sistema in uno stato sicuro. Se si rispetta tale condizione, si assegnano le risorse, altrimenti il processo deve attendere che qualche altro processo ne rilasci un numero sufficiente.

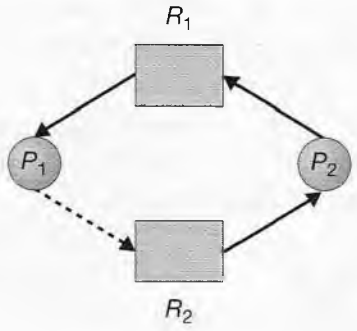

Figura 7.7 Stato non sicuro in un grafo di assegnazione delle risorse.

La realizzazione dell'algoritmo del banchiere richiede la gestione di alcune strutture dati che codificano lo stato di assegnazione delle risorse del sistema. Sia n il numero di processi del sistema e m il numero dei tipi di risorsa. Sono necessarie le seguenti strutture dati.

- **Disponibili.** Un vettore di lunghezza m indica il numero delle istanze disponibili per ciascun tipo di risorsa; *Disponibili*[j] = k, significa che sono disponibili k istanze del tipo di risorsa R_j.
- **Massimo.** Una matrice $n \times m$ definisce la richiesta massima di ciascun processo; *Massimo*[i, j] = k significa che il processo P_i può richiedere un massimo di k istanze del tipo di risorsa R_j.
- **Assegnate.** Una matrice $n \times m$ definisce il numero delle istanze di ciascun tipo di risorsa attualmente assegnate a ogni processo; *Assegnate*[i, j] = k significa che al processo P_i sono correntemente assegnate k istanze del tipo di risorsa R_j.
- **Necessità.** Una matrice $n \times m$ indica la necessità residua di risorse relativa a ogni processo; *Necessità*[i, j] = k significa che il processo P_i, per completare il suo compito, può avere bisogno di altre k istanze del tipo di risorsa R_j. Si osservi che *Necessità*[i, j] = *Massimo*[i, j] – *Assegnate*[i, j].

Col trascorrere del tempo, queste strutture dati variano sia nelle dimensioni sia nei valori.

Per semplificare la presentazione dell'algoritmo del banchiere, si usano le seguenti notazioni: supponendo che X e Y siano vettori di lunghezza n, si può affermare che $X \leq Y$ se e solo se $X[i] \leq Y[i]$ per ogni $i = 1, 2, ..., n$. Ad esempio, se $X = (1, 7, 3, 2)$ e $Y = (0, 3, 2, 1)$, allora $Y \leq X$; $Y < X$ se $Y \leq X$ e $Y \neq X$.

Tutte le righe delle matrici *Assegnate* e *Necessità* sono considerabili vettori e si possono chiamare rispettivamente $Assegnate_i$ e $Necessità_i$. Il vettore $Assegnate_i$ specifica le risorse correntemente assegnate al processo P_i, mentre il vettore $Necessità_i$ specifica le risorse che il processo P_i può ancora richiedere per completare il suo compito.

7.5.3.1 Algoritmo di verifica della sicurezza

L'algoritmo utilizzato per scoprire se il sistema è o non è in uno stato sicuro si può descrivere come segue.

1. Siano *Lavoro* e *Fine* vettori di lunghezza rispettivamente m e n, e inizializza *Lavoro* = *Disponibili* e *Fine*[i] = *falso*, per $i = 1, 2, ..., n - 1$;
2. cerca un indice i tale che valgano contemporaneamente le seguenti relazioni:
   a) *Fine*[i] == *falso*
   b) $Necessità_i \leq Lavoro$

   se tale i non esiste, esegue il passo 4;
3. $Lavoro = Lavoro + Assegnate_i$
   Fine[i] = *vero*
   torna al passo 2
4. se *Fine*[i] == *vero* per ogni i, allora il sistema è in uno stato sicuro.

Per determinare se uno stato è sicuro tale algoritmo può richiedere un numero di operazioni dell'ordine di $m \times n^2$.

7.5.3.2 Algoritmo di richiesta delle risorse

Si descrive ora l'algoritmo che determina se le richieste possano essere garantite in uno stato sicuro.

Sia $Richieste_i$ il vettore delle richieste per il processo P_i. Se $Richieste_i[j] == k$, allora il processo P_i richiede k istanze del tipo di risorsa R_j. Se il processo P_i fa una richiesta di risorse, si svolgono le seguenti azioni:

1. se $Richieste_i \leq Necessità_i$, esegue il passo 2, altrimenti riporta una condizione d'errore, poiché il processo ha superato il numero massimo di richieste;
2. se $Richieste_i \leq Disponibili$, esegue il passo 3, altrimenti P_i deve attendere poiché le risorse non sono disponibili;
3. simula l'assegnazione al processo P_i delle risorse richieste modificando come segue lo stato di assegnazione delle risorse:

   $Disponibili = Disponibili - Richieste_i$

   $Assegnate_i = Assegnate_i + Richieste_i$

   $Necessità_i = Necessità_i - Richieste_i$

   Se lo stato di assegnazione delle risorse risultante è sicuro, la transazione è completata e al processo P_i si assegnano le risorse richieste. Tuttavia, se il nuovo stato è non sicuro, P_i deve attendere $Richieste_i$ e si ripristina il vecchio stato di assegnazione delle risorse.

7.5.3.3 Un esempio

Illustriamo infine l'uso dell'algoritmo del banchiere, considerando un sistema con cinque processi, da P_0 a P_4, e tre tipi di risorse: *A*, *B*, e *C*. Il tipo di risorse *A* ha 10 istanze, il tipo *B* ha 5 istanze e il tipo *C* ha 7 istanze. Si supponga che all'istante T_0 si sia verificata la seguente "istantanea":

	Assegnate	*Massimo*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Il contenuto della matrice *Necessità* è definito come *Massimo* – *Assegnate*:

	Necessità
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Il sistema si trova attualmente in uno stato sicuro; infatti, la sequenza $<P_1, P_3, P_4, P_2, P_0>$ soddisfa i criteri di sicurezza. Si supponga ora che il processo P_1 richieda un'altra istanza del tipo di risorsa A e due istanze del tipo C, quindi $Richieste_1 = (1, 0, 2)$. Per stabilire se questa richiesta si possa esaudire immediatamente si verifica la condizione $Richieste_1 \leq Disponibili$ (vale a dire $(1, 0, 2) \leq (3, 3, 2)$), che risulta vera, quindi, supponendo che questa richiesta sia stata soddisfatta, si ottiene il seguente nuovo stato:

	Assegnate	*Necessità*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Occorre stabilire se questo nuovo stato del sistema sia sicuro; a tale scopo si esegue l'algoritmo di verifica della sicurezza da cui risulta che la sequenza $<P_1, P_3, P_4, P_0, P_2>$ rispetta il requisito di sicurezza. Quindi si può soddisfare immediatamente la richiesta del processo P_1.

Tuttavia, dovrebbe essere chiaro che, quando il sistema si trova in questo stato, una richiesta di (3, 3, 0) da parte di P_4 non si può soddisfare finché non siano disponibili le risorse. Inoltre, una richiesta di (0, 2, 0) da parte di P_0 non si potrebbe soddisfare neanche se le risorse fossero disponibili, poiché lo stato risultante sarebbe non sicuro. L'implementazione dell'algoritmo del banchiere è lasciata come esercizio di programmazione.

7.6 Rilevamento delle situazioni di stallo

Se un sistema non si avvale di un algoritmo per prevenire o evitare situazioni di stallo, è possibile che si verifichi effettivamente. In tal caso il sistema deve fornire i seguenti algoritmi:

- un algoritmo che esamini lo stato del sistema per stabilire se si è verificato uno stallo;
- un algoritmo che ripristini il sistema dalla condizione di stallo.

Nell'analisi seguente sono trattati i suddetti argomenti che riguardano sia sistemi con una sola istanza di ciascun tipo di risorsa, sia sistemi con più istanze. Tuttavia, a questo punto, occorre notare che uno schema di rilevamento e ripristino richiede un carico che include sia i costi operativi per la memorizzazione delle informazioni necessarie e per l'esecuzione dell'algoritmo di rilevamento, sia i potenziali costi connessi al ripristino da una situazione di stallo.

7.6.1 Istanza singola di ciascun tipo di risorsa

Se tutte le risorse hanno una sola istanza si può definire un algoritmo di rilevamento di situazioni di stallo che fa uso di una variante del grafo di assegnazione delle risorse, detta **grafo d'attesa**, ottenuta dal grafo di assegnazione delle risorse togliendo i nodi dei tipi di risorse e componendo gli archi tra i processi.

Più precisamente, un arco da P_i a P_j del grafo d'attesa implica che il processo P_i attende che il processo P_j rilasci una risorsa di cui P_i ha bisogno. Un arco $P_i \rightarrow P_j$ esiste nel grafo d'attesa se e solo se il corrispondente grafo di assegnazione delle risorse contiene due archi $P_i \rightarrow R_q$ e $R_q \rightarrow P_j$ per qualche risorsa R_q. Nella Figura 7.8 sono illustrati un grafo di assegnazione delle risorse e il corrispondente grafo d'attesa.

Come in precedenza, nel sistema esiste uno stallo se e solo se il grafo d'attesa contiene un ciclo. Per individuare le situazioni di stallo, il sistema deve *conservare* il grafo d'attesa e *invocare periodicamente un algoritmo* che cerchi un ciclo all'interno del grafo. L'algoritmo per il rilevamento di un ciclo all'interno di un grafo richiede un numero di operazioni dell'ordine di n^2, dove con n si indica il numero dei vertici del grafo.

7.6.2 Più istanze di ciascun tipo di risorsa

Lo schema con grafo d'attesa non si può applicare ai sistemi di assegnazione delle risorse con più istanze di ciascun tipo di risorsa. Il seguente algoritmo di rilevamento di situazioni di stallo è, invece, applicabile a tali sistemi. Esso si serve di strutture dati variabili nel tempo, simili a quelle adoperate nell'algoritmo del banchiere (Paragrafo 7.5.3).

- *Disponibili*. Vettore di lunghezza m che indica il numero delle istanze disponibili per ciascun tipo di risorsa.
- *Assegnate*. Matrice $n \times m$ che definisce il numero delle istanze di ciascun tipo di risorse correntemente assegnate a ciascun processo.
- *Richieste*. Matrice $n \times m$ che indica la richiesta attuale di ciascun processo. Se *Richieste*$[i, j] = k$, significa che il processo P_i sta richiedendo altre k istanze del tipo di risorsa R_j.

La relazione $\leq$ tra due vettori si definisce come nel Paragrafo 7.5.3. Per semplificare la notazione, le righe delle matrici *Assegnate* e *Richieste* si trattano come vettori e, nel seguito, sono indicate rispettivamente come *Assegnate*$_i$ e *Richieste*$_i$. L'algoritmo di rilevamento descritto indaga su ogni possibile sequenza di assegnazione per i processi che devono ancora essere completati. Questo algoritmo si può confrontare con quello del del banchiere del Paragrafo 7.5.3.

1. Siano *Lavoro* e *Fine* vettori di lunghezza rispettivamente m e n, inizializza *Lavoro* = *Disponibili*, per $i = 1, 2, \ldots, n$, se *Assegnate*$_i \neq 0$, allora *Fine*$[i]$ = *falso*, altrimenti *Fine*$[i]$ = *vero*;
2. cerca un indice i tale che valgano contemporaneamente le seguenti relazioni:
   a) *Fine*$[i]$ == *falso*
   b) *Richieste*$_i \leq$ *Lavoro*

   se tale i non esiste, esegue il passo 4;
3. *Lavoro* = *Lavoro* + *Assegnate*$_i$
   Fine$[i]$ = *vero*
   torna al passo 2
4. se *Fine*$[i]$ == *falso* per qualche i, $0 \leq i < n$, allora il sistema è in stallo, inoltre, se *Fine*$[i]$ == *falso*, il processo P_i è in stallo.

Tale algoritmo richiede un numero di operazioni dell'ordine di $m \times n^2$ per controllare se il sistema è in stallo.

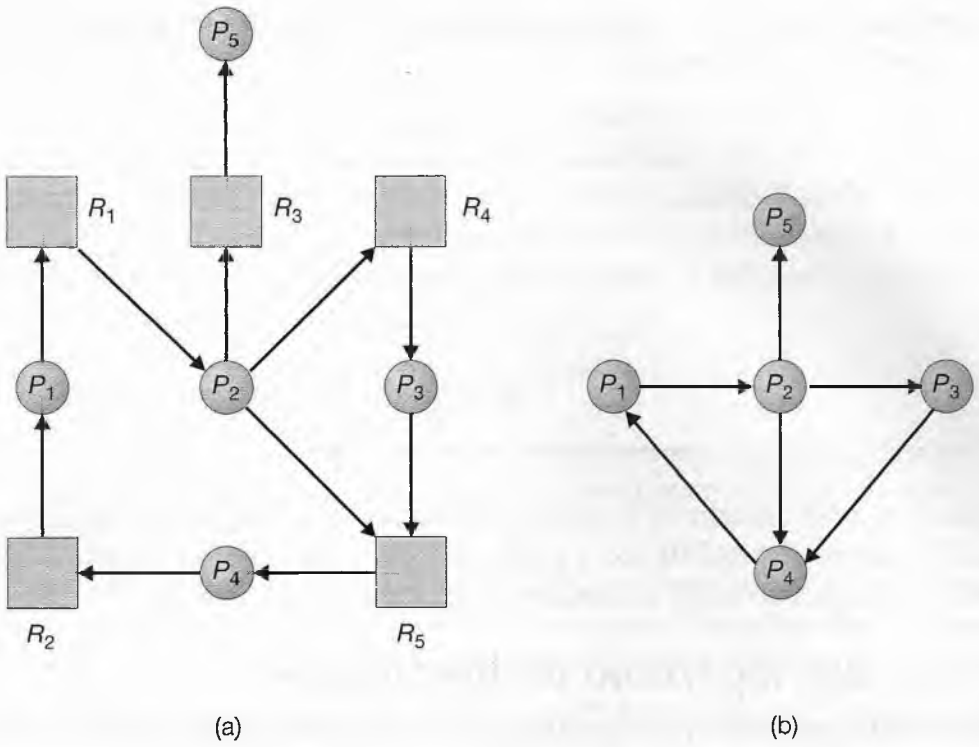

Figura 7.8 (a) Grafo di assegnazione delle risorse; (b) grafo d'attesa corrispondente.

Può meravigliare che le risorse del processo P_i siano richieste (passo 3) non appena risulta valida la condizione *Richieste*$_i \leq$ *Lavoro* (passo 2.b). Tale condizione garantisce che P_i *non* è correntemente coinvolto in uno stallo, quindi, assumendo un atteggiamento ottimistico, si suppone che P_i non intenda richiedere altre risorse per completare il proprio compito, e che restituisca presto tutte le risorse. Se non si rispetta l'ipotesi fatta, si può verificare uno stallo, che sarà rilevato quando si richiamerà nuovamente l'algoritmo di rilevamento.

Per illustrare questo algoritmo, si consideri un sistema con cinque processi, da P_0 a P_4, e tre tipi di risorse: *A*, *B*, e *C*. Il tipo di risorsa *A* ha 7 istanze, il tipo *B* ha 2 istanze e il tipo *C* ne ha 6. Si supponga di avere, all'istante T_0, il seguente stato di assegnazione delle risorse:

	Assegnate	*Richieste*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Il sistema non è in stallo. Infatti eseguendo l'algoritmo per la sequenza <P_0, P_2, P_3, P_1, P_4>, risulta *Fine*[*i*] == *vero* per ogni *i*.

Si supponga ora che il processo P_2 richieda un'altra istanza di tipo *C*. La matrice *Richieste* viene modificata come segue:

	Richieste
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

Ora il sistema è in stallo. Anche se si possono reclamare le risorse possedute dal processo P_0, il numero delle risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi, quindi si verifica uno stallo composto dai processi P_1, P_2, P_3 e P_4.

7.6.3 Uso dell'algoritmo di rilevamento

Per sapere quando è necessario ricorrere all'algoritmo di rilevamento si devono considerare i seguenti fattori:

1. *frequenza* (presunta) con la quale si verifica uno stallo;
2. *numero* dei processi che sarebbero influenzati da tale stallo.

Se le situazioni di stallo sono frequenti, è necessario ricorrere spesso all'algoritmo per il loro rilevamento. Le risorse assegnate a processi in stallo rimangono inattive fino all'eliminazione dello stallo. Inoltre, il numero dei processi coinvolti nel ciclo di stallo può aumentare.

Le situazioni di stallo si verificano solo quando qualche processo fa una richiesta che non si può soddisfare immediatamente; può essere una richiesta che chiude una catena di processi in attesa. Il caso estremo è quello nel quale l'algoritmo di rilevamento si usa ogni volta che non si può soddisfare immediatamente una richiesta di assegnazione. In questo caso non si identifica soltanto il gruppo di processi in stallo, ma anche il processo che ha "causato" lo stallo, anche se, in verità, ciascuno dei processi in stallo è un elemento del ciclo all'interno del grafo di assegnazione delle risorse, quindi tutti i processi sono, congiuntamente, responsabili dello stallo. Se esistono tipi di risorsa diversi, una singola richiesta può causare più cicli nel grafo delle risorse, ciascuno dei quali è completato dalla richiesta più recente ed è causato da un processo identificabile.

Naturalmente, l'uso dell'algoritmo di rilevamento per ogni richiesta aumenta notevolmente il carico nei termini di tempo di calcolo. Un'alternativa meno dispendiosa è quella in cui l'algoritmo di rilevamento s'invoca a intervalli meno frequenti, per esempio una volta ogni ora, oppure ogni volta che l'utilizzo della CPU scende sotto il 40 per cento, poiché uno stallo può rendere inefficienti le prestazioni del sistema e quindi causare una drastica riduzione dell'utilizzo della CPU. Non è conveniente richiedere l'algoritmo di rilevamento in momenti arbitrari, poiché nel grafo delle risorse possono coesistere molti cicli e, normalmente, non si può dire quale fra i tanti processi in stallo abbia "causato" lo stallo.

7.7 Ripristino da situazioni di stallo

Una situazione di stallo si può affrontare in diversi modi. Una soluzione consiste nell'informare l'operatore della presenza dello stallo, in modo che possa gestirlo manualmente. L'altra soluzione lascia al sistema il ripristino automatico dalla situazione di stallo. Uno stallo si può eliminare in due modi: il primo prevede semplicemente la terminazione di uno o più processi per interrompere l'attesa circolare; il secondo esercita la prelazione su alcune risorse in possesso di uno o più processi in stallo.

7.7.1 Terminazione di processi

Per eliminare le situazioni di stallo attraverso la terminazione di processi si possono adoperare due metodi; in entrambi il sistema recupera immediatamente tutte le risorse assegnate ai processi terminati.

- **Terminazione di tutti i processi in stallo.** Chiaramente questo metodo interrompe il ciclo di stallo, ma l'operazione è molto onerosa; questi processi possono aver già fatto molti calcoli i cui risultati si annullano e probabilmente dovranno essere ricalcolati.
- **Terminazione di un processo alla volta fino all'eliminazione del ciclo di stallo.** Questo metodo causa un notevole carico, poiché, dopo aver terminato ogni processo, si deve impiegare un algoritmo di rilevamento per stabilire se esistono ancora processi in stallo.

Procurare la terminazione di un processo può essere un'operazione tutt'altro che semplice: se il processo si trova nel mezzo dell'aggiornamento di un file, la terminazione lascia il file in uno stato scorretto; analogamente, se il processo si trova nel mezzo di una stampa di dati, prima di stampare un lavoro successivo, il sistema deve reimpostare la stampante riportandola a uno stato corretto.

Se si adopera il metodo di terminazione parziale, dato un insieme di processi in stallo occorre determinare quale processo, o quali processi, costringere alla terminazione nel tentativo di sciogliere la situazione di stallo. Analogamente ai problemi di scheduling della CPU, si tratta di scegliere un criterio. Le considerazioni sono essenzialmente economiche: si dovrebbero arrestare i processi la cui terminazione causa il minimo costo. Sfortunatamente, il termine *minimo costo* non è preciso. La scelta dei processi è influenzata da diversi fattori, tra cui i seguenti:

1. la priorità dei processi;
2. il tempo trascorso dalla computazione e il tempo ancora necessario per completare i compiti assegnati ai processi;
3. la quantità e il tipo di risorse impiegate dai processi (ad esempio, se si può avere facilmente la prelazione sulle risorse);
4. la quantità di ulteriori risorse di cui i processi hanno ancora bisogno per completare i propri compiti;
5. il numero di processi che si devono terminare;
6. il tipo di processi: interattivi o a lotti.

7.7.2 Prelazione su risorse

Per eliminare uno stallo si può esercitare la prelazione sulle risorse: le risorse si sottraggono in successione ad alcuni processi e si assegnano ad altri finché si ottiene l'interruzione del ciclo di stallo.

Se per gestire le situazioni di stallo s'impiega la prelazione, si devono considerare i seguenti problemi.

1. **Selezione di una vittima.** Occorre stabilire quali risorse e quali processi si devono sottoporre a prelazione. Come per la terminazione dei processi, è necessario stabilire l'ordine di prelazione allo scopo di minimizzare i costi. I fattori di costo possono includere parametri come il numero delle risorse possedute da un processo in stallo, e la quantità di tempo già spesa durante l'esecuzione da un processo in stallo.
2. **Ristabilimento di un precedente stato sicuro.** Occorre stabilire che cosa fare con un processo cui è stata sottratta una risorsa. Poiché è stato privato di una risorsa necessaria, la sua esecuzione non può continuare normalmente, quindi deve essere ricondotto a un precedente stato sicuro dal quale essere riavviato. Poiché, in generale, è difficile stabilire quale stato sia sicuro, la soluzione più semplice consiste nel terminare il processo e quindi riavviarlo. Certamente, è più efficace ristabilire un precedente stato sicuro del processo che sia sufficiente allo scioglimento della situazione di stallo, ma questo metodo richiede che il sistema mantenga più informazioni sullo stato di tutti i processi in esecuzione.
3. **Attesa indefinita.** È necessario assicurare che non si verifichino situazioni d'attesa indefinita, occorre cioè garantire che le risorse non siano sottratte sempre allo stesso processo.

   In un sistema in cui la scelta della vittima avviene soprattutto secondo fattori di costo, può accadere che si scelga sempre lo stesso processo; in questo caso il processo non riesce mai a completare il suo compito; si tratta di una situazione d'attesa indefinita che si deve affrontare in qualsiasi sistema concreto. Chiaramente è necessario assicurare che un processo possa essere prescelto come vittima solo un numero finito (e ridotto) di volte; la soluzione più diffusa prevede l'inclusione del numero di terminazioni, per successivi riavvii, tra i fattori di costo.

7.8 Sommario

Uno stallo si verifica quando in un insieme di processi ciascun processo dell'insieme attende un evento che può essere causato solo da un altro processo dell'insieme. In linea di principio, i metodi di gestione delle situazioni di stallo sono tre:

- impiegare un protocollo che prevenga o eviti le situazioni di stallo, assicurando che il sistema non vi entri mai;
- permettere al sistema di entrare in stallo e quindi effettuarne il ripristino;
- ignorare del tutto il problema fingendo che nel sistema non si verifichino mai situazioni di stallo. Tale "soluzione" è adottata dalla maggior parte dei sistemi operativi, compresi UNIX e Windows.

Una situazione di stallo può presentarsi solo se all'interno del sistema si verificano contemporaneamente quattro condizioni necessarie: *mutua esclusione*, *possesso e attesa*, *impossibilità di prelazione* e *attesa circolare*. Per prevenire il verificarsi di situazioni di stallo è necessario assicurare che almeno una delle suddette condizioni necessarie non sia soddisfatta.

Un altro metodo per evitare le situazioni di stallo, meno rigido dell'algoritmo di prevenzione, consiste nel disporre di informazioni a priori su come ciascun processo intende impiegare le risorse. L'algoritmo del banchiere, per esempio, richiede la conoscenza del numero massimo di ogni classe di risorse che può essere richiesto da ciascun processo. Servendosi di queste informazioni si può definire un algoritmo per evitare le situazioni di stallo.

Se un sistema non usa alcun protocollo per assicurare che non avvengano situazioni di stallo, è necessario un metodo di rilevamento e ripristino. Per stabilire se si sia verificato uno stallo, è necessario ricorrere a un algoritmo di rilevamento; nel caso in cui si rilevi uno stallo, il sistema deve attuare il ripristino terminando alcuni processi coinvolti, oppure sottraendo risorse a qualcuno di essi.

Nel caso in cui la prelazione sia applicata alla prevenzione dello stallo, occorre tenere conto di tre questioni: la selezione di una vittima, il ristabilimento di un precedente stato sicuro e l'attesa indefinita. Nei sistemi che selezionano le vittime principalmente sulla base di fattori di costo, possono presentarsi situazioni di attesa indefinita, causando l'impossibilità del processo scelto di portare a termine il suo compito.

Infine, è stato sostenuto da parte di alcuni ricercatori che nessuno di questi approcci sia in grado, da solo, di risolvere appropriatamente l'intera gamma di problemi legati all'allocazione delle risorse nei sistemi operativi. Ciononostante, questi metodi di base possono essere combinati in modo da fornire strategie ottimali per ogni classe di risorse dei sistemi.

Esercizi pratici

7.1 Elencate tre esempi di stallo di processi che non siano correlati all'informatica.

7.2 Supponete che un sistema sia in uno stato non sicuro. Mostrate che è possibile che i processi completino l'esecuzione senza entrare in una situazione di stallo.

7.3 Un metodo possibile per prevenire gli stalli consiste nel disporre di una singola risorsa di ordine più elevato che deve essere richiesta prima di ogni altra risorsa. Lo stallo è possibile, ad esempio, se i thread multipli tentano di accedere agli oggetti di sincronizzazione *A*···*E*. (Tali oggetti di sincronizzazione possono includere mutex, semafori, variabili condition, e simili.) Possiamo prevenire gli stalli aggiungendo un sesto oggetto *F*. Ogni volta che un thread vuole acquisire un lock di sincronizzazione per ciascun oggetto *A*···*E*, deve prima acquisire il lock per l'oggetto *F*. Questa soluzione è conosciuta come **inclusione** *(containment)*: i lock per gli oggetti *A*···*E* sono inclusi all'interno del lock per l'oggetto *F*. Paragonate questo schema con quello ad attesa circolare del Paragrafo 7.4.4.

7.4 Dimostrate che l'algoritmo di sicurezza presentato nel Paragrafo 7.5.3 richieda un numero di operazioni dell'ordine di $m \times n^2$.

7.5 Considerate un sistema informatico che esegua 5000 task al mese e non disponga né di uno schema per prevenire gli stalli né di uno schema per evitarli. Gli stalli si verificano circa due volte al mese e l'operatore deve terminare e rieseguire circa 10 task per ogni stallo. Ogni task costa circa 2 dollari (in tempo del processore), e i task terminati tendono a essere circa a metà del loro lavoro quando vengono interrotti.

Un programmatore di sistema ha stimato che un algoritmo per evitare gli stalli (come l'algoritmo del banchiere) potrebbe essere installato nel sistema con un incremento del 10% circa del tempo medio di esecuzione per task. Siccome la macchina ha attualmente il 30% di periodo d'inattività, tutti i 5000 task al mese potrebbero ancora essere eseguiti, nonostante il tempo di completamento aumenterebbe in media del 20 per cento circa.

a. Quali sono i vantaggi dell'installazione dell'algoritmo per evitare gli stalli?
b. Quali sono gli svantaggi dell'installazione dell'algoritmo per evitare gli stalli?

7.6 Un sistema può individuare quali dei suoi processi stanno morendo? Se la vostra risposta è affermativa, spiegate com'è possibile. Se è negativa, spiegate come il sistema può trattare il problema dell'attesa indefinita.

7.7 Considerate la seguente politica di allocazione delle risorse. Le richieste e i rilasci di risorse sono sempre possibili. Se una richiesta di risorse non può essere soddisfatta perché queste non sono disponibili, si controllano tutti i processi bloccati in attesa di risorse. Se un processo bloccato possiede le risorse desiderate, queste possono essere prelevate perché vengano date al processo che le richiede. Il vettore delle risorse attese dal processo bloccato è aggiornato in modo da includere le risorse sottratte al processo.

Si consideri ad esempio un sistema con tre tipi di risorse e il vettore *Disponibili* inizializzato a (4,2,2). Se il processo P_0 richiede (2,2,1), le ottiene. Se P_1 richiede (1,0,1), le ottiene. Poi, se P_0 chiede (0,0,1) viene bloccato (risorsa non disponibile). Se P_2 ora chiede (2,0,0) ottiene la risorsa disponibile (1,0,0) e quella che è stata assegnata a P_0 (poiché P_0 è bloccato). Il vettore *Assegnate* di P_0 scende a (1,2,1) e il suo vettore *Necessità* passa a (1,0,1).

a. Possono verificarsi stalli? Se la risposta è affermativa, fornite un esempio. In caso di risposta negativa, specificate quale condizione necessaria non si può verificare.
b. Può verificarsi un blocco indefinito? Spiegate la risposta.

7.8 Supponete di aver codificato l'algoritmo di sicurezza per evitare gli stalli e che ora vi sia richiesto di implementare l'algoritmo di rilevamento delle situazioni di stallo. È possibile farlo utilizzando semplicemente il codice dell'algoritmo di sicurezza e ridefinendo $Massimo_i = Attesa_i + Assegnate_i$, dove $Attesa_i$ è un vettore che specifica le risorse attese dal processo i e $Assegnate_i$ è definito come nel Paragrafo 7.5? Motivate la risposta.

7.9 È possibile che uno stallo coinvolga un solo processo? Argomentate la risposta.

Esercizi

7.10 Considerate lo stallo di traffico automobilistico illustrato nella Figura 7.9.

a. dimostrate che le quattro condizioni necessarie per lo stallo valgono anche in questo esempio;
b. fissate una semplice regola che eviti le situazioni di stallo in questo sistema.

7.11 Considerate la situazione di stallo che potrebbe verificarsi con il problema dei cinque filosofi, allorquando essi ricevano le bacchette, una per volta. Chiarite come le quat-

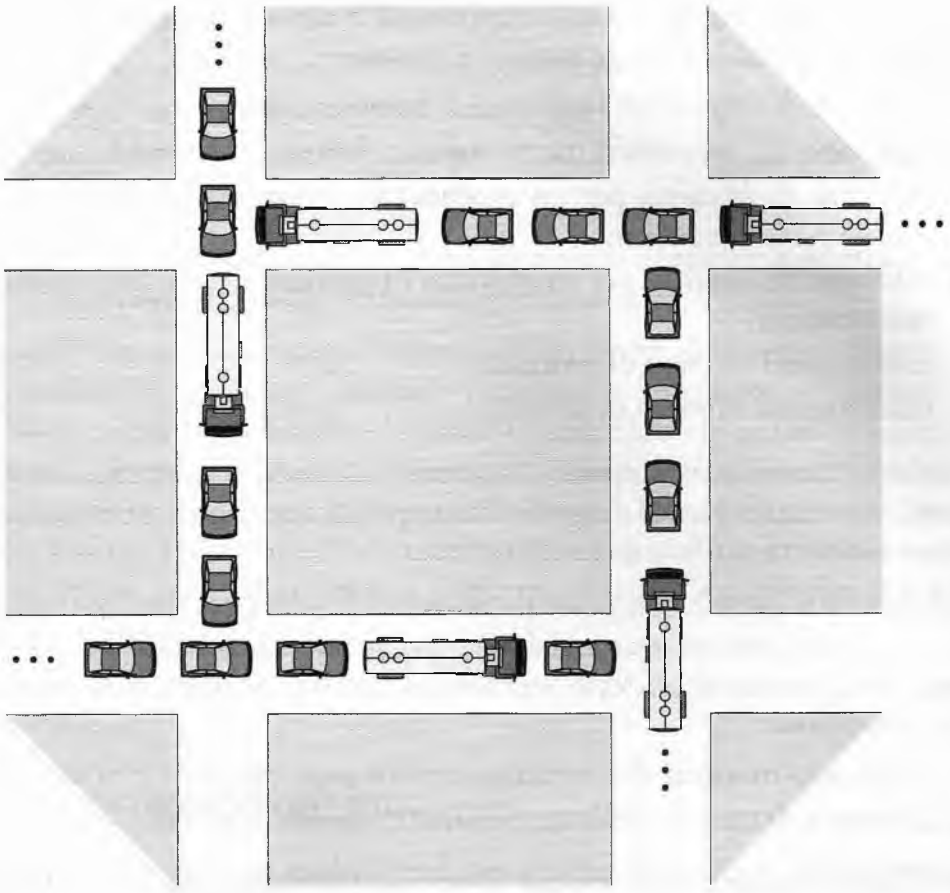

Figura 7.9 Stallo di traffico automobilistico per l'Esercizio 7.10.

tro condizioni necessarie per lo stallo abbiano validità in questo contesto. Discutete come, eliminando una delle quattro condizioni, lo stallo potrebbe essere evitato.

7.12 Un rimedio al pericolo dello stallo è rappresentato dalla disponibilità di una sola risorsa, di grado superiore, che deve essere richiesta prima di qualunque altra. Se, a esempio, svariati thread tentano di accedere agli oggetti *A*, ..., *E*, lo stallo è possibile. (Tali oggetti possono includere mutex, semafori, variabili condizionali, e così via.) Ma con l'aggiunta di un sesto oggetto *F*, possiamo neutralizzare il pericolo: ogniqualvolta un thread intenda acquisire il lock che regola la sincronizzazione di un oggetto *A*, ..., *E*, esso dovrà prima impossessarsi del lock relativo a *F*. Questa soluzione è nota come **contenimento**. I lock relativi agli oggetti *A*, ..., *E* sono "contenuti" nel lock dell'oggetto *F*. Confrontate questo modello con quello dell'attesa circolare, descritto nel Paragrafo 7.4.4.

7.13 Confrontate il modello dell'attesa circolare con le varie strategie di neutralizzazione dello stallo (quali l'algoritmo del banchiere) rispetto alle seguenti problematiche:

a. carico di esecuzione;
b. produttività del sistema.

7.14 In un sistema reale, né le risorse disponibili, né le richieste di risorse da parte dei processi sono coerenti in lunghi periodi (mesi). Le risorse si guastano o sono sostituite, nuovi processi vanno e vengono, si acquistano e si aggiungono nuove risorse al sistema. Se le situazioni di stallo si controllano con l'algoritmo del banchiere, dite quali

tra le seguenti modifiche si possono apportate con sicurezza, senza introdurre la possibilità di situazioni di stallo, e in quali circostanze:

a. aumento di *Disponibili* (aggiunta di nuove risorse);
b. riduzione di *Disponibili* (risorsa rimossa definitivamente dal sistema);
c. aumento di *Massimo* per un processo (il processo necessita di più risorse di quante siano permesse);
d. riduzione di *Massimo* per un processo (il processo decide che non ha bisogno di tante risorse);
e. aumento del numero di processi;
f. riduzione del numero di processi.

7.15 Considerate un sistema composto da quattro risorse dello stesso tipo condivise da tre processi, ciascuno dei quali necessita di non più di due risorse. Dimostrate che non si possono verificare situazioni di stallo.

7.16 Considerate un sistema composto da m risorse dello stesso tipo, condivise da n processi. Le risorse possono essere richieste e rilasciate dai processi solo una alla volta. Dimostrate che non si possono verificare situazioni di stallo se si rispettano le seguenti condizioni:

a. la richiesta massima di ciascun processo è compresa tra 1 e m risorse;
b. la somma di tutte le richieste massime è minore di $m + n$.

7.17 Considerate il problema dei cinque filosofi, supponendo che le bacchette siano disposte al centro del tavolo e che ciascuna coppia di bacchette possa essere usata da un filosofo. Supponete che le richieste di bacchette avvengano una per volta. Descrivete una semplice regola che determini se una certa richiesta possa essere soddisfatta senza causare stallo, data la distribuzione corrente delle bacchette tra i filosofi.

7.18 Considerate lo stesso contesto del problema precedente. Ipotizzate, ora, che ogni filosofo richieda tre bacchette per mangiare e che, anche qui, le richieste siano avanzate separatamente. Descrivete delle semplici regole che determinino se una certa richiesta possa essere soddisfatta senza causare stallo, data la distribuzione corrente delle bacchette tra i filosofi.

7.19 Potete ricavare un algoritmo del banchiere che sia adatto a un solo tipo di risorsa dall'algoritmo generale del banchiere, semplicemente riducendo la dimensione dei vari array di 1. Dimostrate con un esempio che l'algoritmo del banchiere generale non può essere implementato applicando individualmente a ciascun tipo di risorsa l'algoritmo per un solo tipo di risorsa.

7.20 Considerate la seguente "istantanea" di un sistema:

	Assegnate	*Massimo*	*Disponibili*
	A B C D	*A B C D*	*A B C D*
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

Impiegando l'algoritmo del banchiere rispondete alle seguenti domande:

a. dite qual è il contenuto della matrice *Necessità*;
b. dite se il sistema è in uno stato sicuro;
c. dite se a fronte di una richiesta per (0, 4, 2, 0), fatta dal processo P_1, essa può essere soddisfatta immediatamente.

7.21 Quale ottimistica ipotesi è alla base dell'algoritmo di rilevamento delle situazioni di stallo? Come potrebbe essere invalidata questa ipotesi?

7.22 Un ponte a corsia unica collega i due villaggi di North Tunbridge e South Tunbridge, nel Vermont. Gli agricoltori dei due villaggi usano tale ponte per portare i loro prodotti nella cittadina confinante. Il ponte rimane bloccato se un contadino diretto a nord e uno diretto a sud vi salgono contemporaneamente. (I contadini del Vermont sono ostinati: non farebbero mai marcia indietro.) Progettate un algoritmo che eviti lo stallo tramite i semafori. Non curatevi, all'inizio, dell'attesa indefinita, ossia la situazione in cui i contadini diretti a nord impediscono a quelli diretti a sud di salire sul ponte, o viceversa.

7.23 Modificate la soluzione al problema dell'Esercizio 7.22 in modo da eliminare l'attesa indefinita.

Problemi di programmazione

7.24 Elaborate un programma multithread che implementi l'algoritmo del banchiere illustrato nel Paragrafo 7.5.3. Create *n* thread che richiedano alla banca le risorse, e le disimpegnino. Il banchiere soddisferà la richiesta solo se ciò può avvenire lasciando il sistema in uno stato sicuro. Potete scegliere, per il programma, i thread di Pthreads o di Win32. È importante che l'accesso ai dati condivisi sia protetto dall'accesso concorrente; a tale scopo, si possono impiegare i lock mutex, disponibili sia nella API di Pthreads sia in quella di Win32. L'argomento, per entrambe queste librerie, è trattato nel progetto alla fine del Capitolo 6.

7.9 Note bibliografiche

[Dijkstra 1965a] è stato uno dei primi e più influenti studiosi del fenomeno dello stallo. [Holt 1972] è stato il primo a formalizzare la nozione di stallo tramite un modello basato sulla teoria dei grafi simile a quello presentato in questo capitolo; nello stesso lavoro, si è inoltre occupato dell'attesa indefinita. [Hyman 1985] presenta l'esempio di situazione di stallo tratto dalla legislazione del Kansas. Uno studio recente sulla gestione dello stallo è fornito da [Levine 2003].

I diversi algoritmi di prevenzione sono stati suggeriti da [Havender 1968], che ha progettato lo schema di ordinamento delle risorse per il sistema IBM OS/360.

L'algoritmo del banchiere è stato sviluppato per un singolo tipo di risorsa da [Dijkstra 1965a], ed è stato esteso a più tipi di risorse da [Habermann 1969]. Gli Esercizi 7.15 e 7.16 sono tratti da [Holt 1971].

L'algoritmo di rilevamento delle situazioni di stallo per istanze multiple di un tipo di risorsa, descritto nel Paragrafo 7.6.2, è presentato in [Coffman et al. 1971].