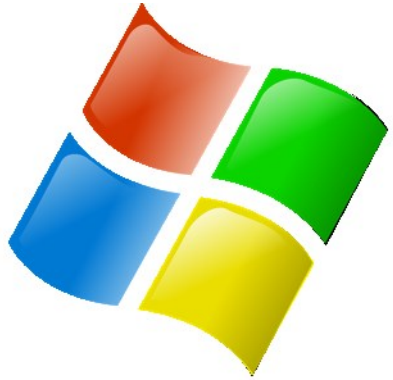
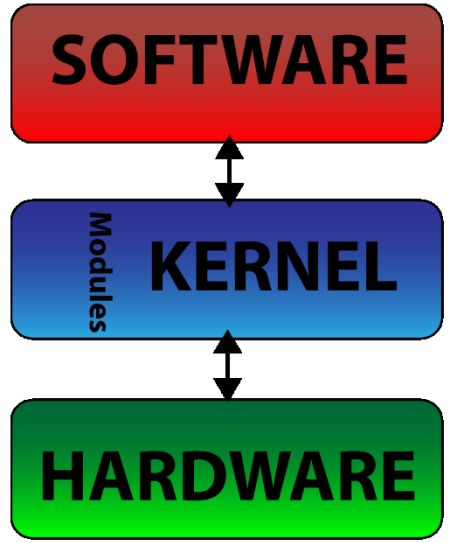
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA



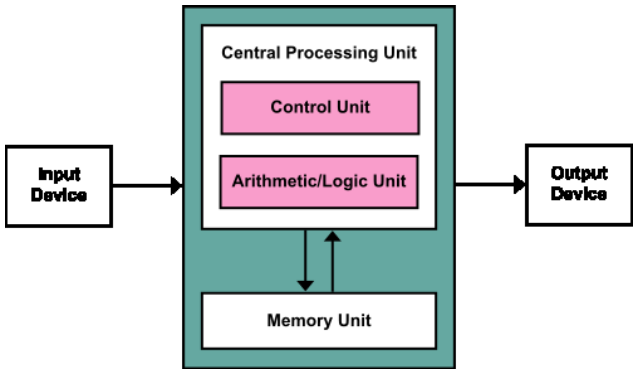
Corso di Sistemi Operativi
A.A. 2019/20

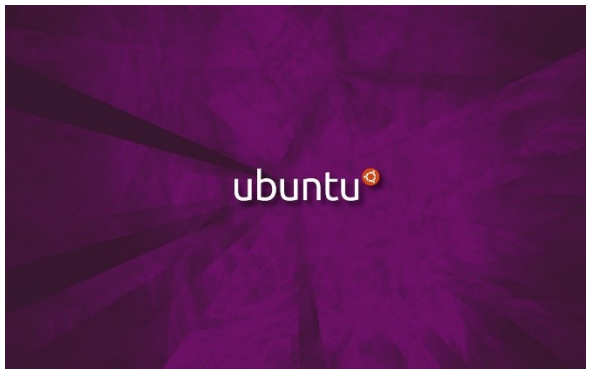
Stallo dei processi



Docente:
Domenico Daniele Bloisi

Novembre 2019

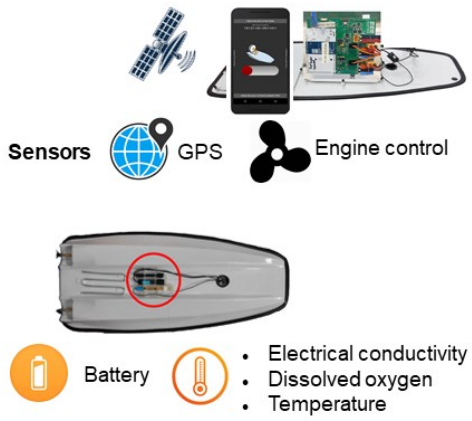


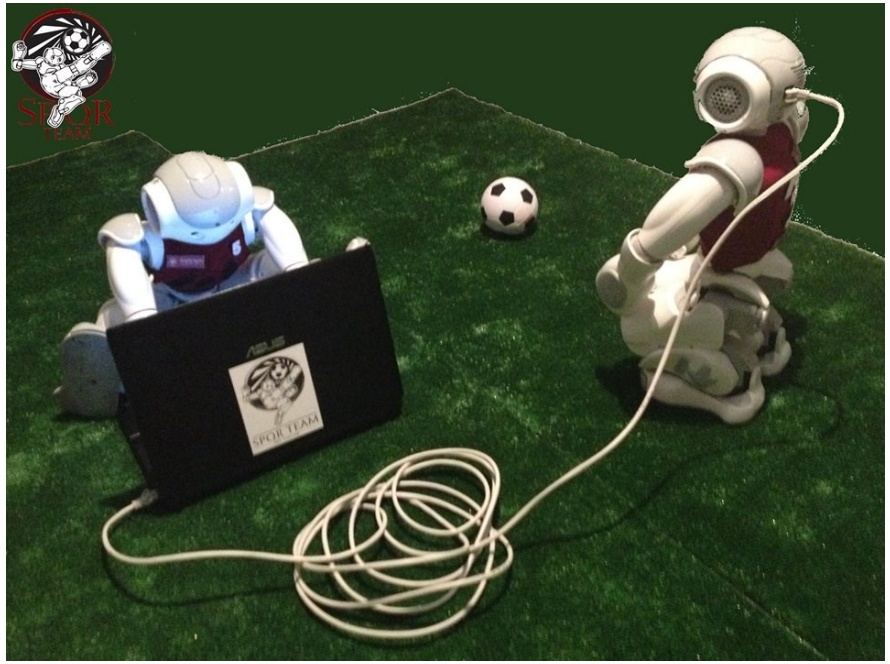


Domenico Daniele Bloisi

- Ricercatore RTD B
  Dipartimento di Matematica, Informatica ed Economia
  Università degli studi della Basilicata
  http://web.unibas.it/bloisi

- SPQR Robot Soccer Team
  Dipartimento di Informatica, Automatica e Gestionale Università degli studi di Roma “La Sapienza”
  http://spqr.diag.uniroma1.it





Ricevimento

- In aula, subito dopo le lezioni
- Martedì dalle 11:00 alle 13:00 presso:
  Campus di Macchia Romana
  Edificio 3D (Dipartimento di Matematica, Informatica ed Economia)
  II piano, stanza 15

Email: domenico.bloisi@unibas.it

Programma – Sistemi Operativi

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

Risorse

Un **sistema** è composto da un numero finito di risorse.

In un ambiente con multiprogrammazione più thread possono competere per ottenere tali risorse

Un thread può servirsi di una risorsa soltanto se rispetta la seguente sequenza:

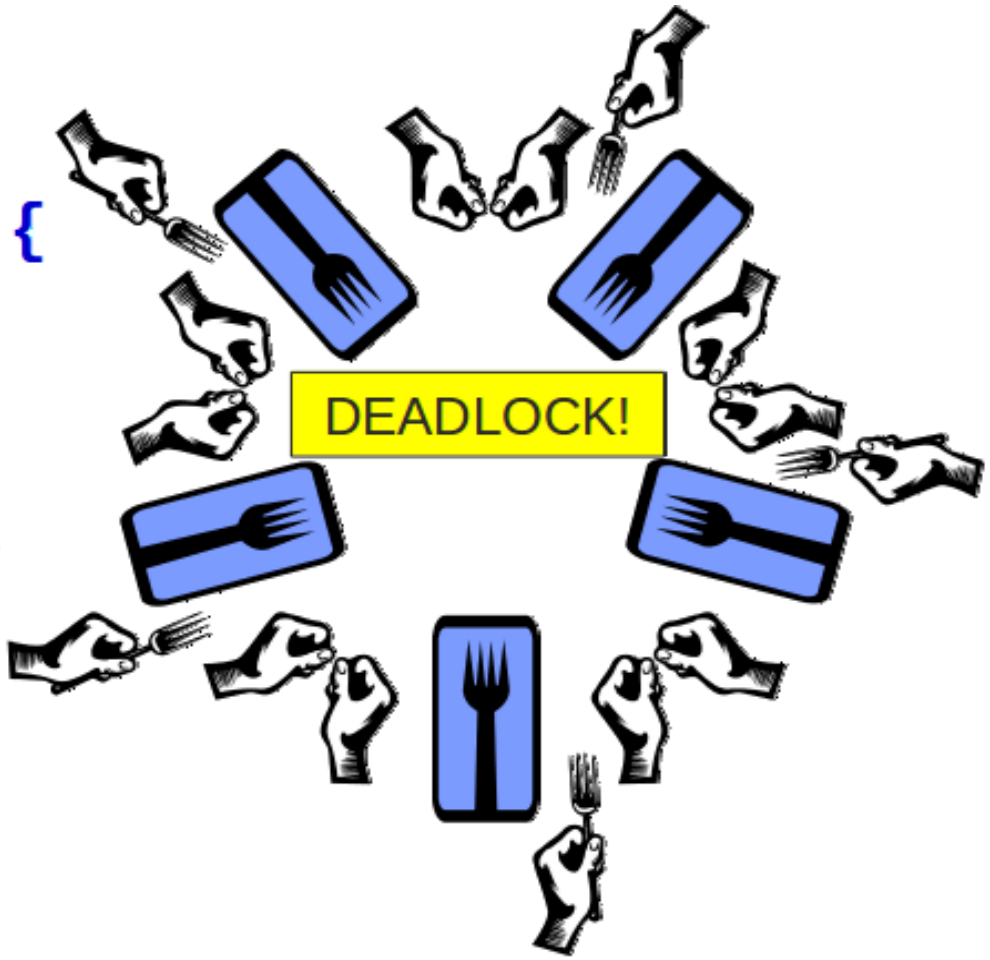
Stallo

Se le risorse richieste da un thread T sono trattenute da altri thread, a loro volta nello stato di attesa, il thread T potrebbe non cambiare più il suo stato.

Situazioni di questo tipo sono chiamate di stallo (deadlock)

Esempio di stallo

```
# define N 5

void philosopher (int i) {
    while (TRUE) {
        think();
        take_fork(i);
        take_fork((i+1)%N);
        eat(); /* yummy */
        put_fork(i);
        put_fork((i+1)%N);
    }
}
```



https://courses.engr.illinois.edu/cs241/fa2011/lectures/final/17-ClassicSynch.ppt

Esempio di stallo in applicazioni multithread

```
/* thread_one esegue in questa funzione */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /**
      * Fa qualcosa
      */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);

   pthread_exit(0);
}

/* thread_two esegue in questa funzione */
void *do_work_two(void *param)
{
   pthread_mutex_lock (&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /**
      * Fa qualcosa
      */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);

   pthread_exit(0);
}
```

Figura 8.1 Esempio di stallo.

Stallo attivo (livelock)

Lo stallo attivo o livelock si verifica quando un thread tenta continuamente un'azione che non ha successo.

Il livelock è meno comune del deadlock, ma è comunque un problema complesso nella progettazione di applicazioni concorrenti e, come il deadlock, può verificarsi solo in determinate condizioni di scheduling.

Stallo attivo (livelock)

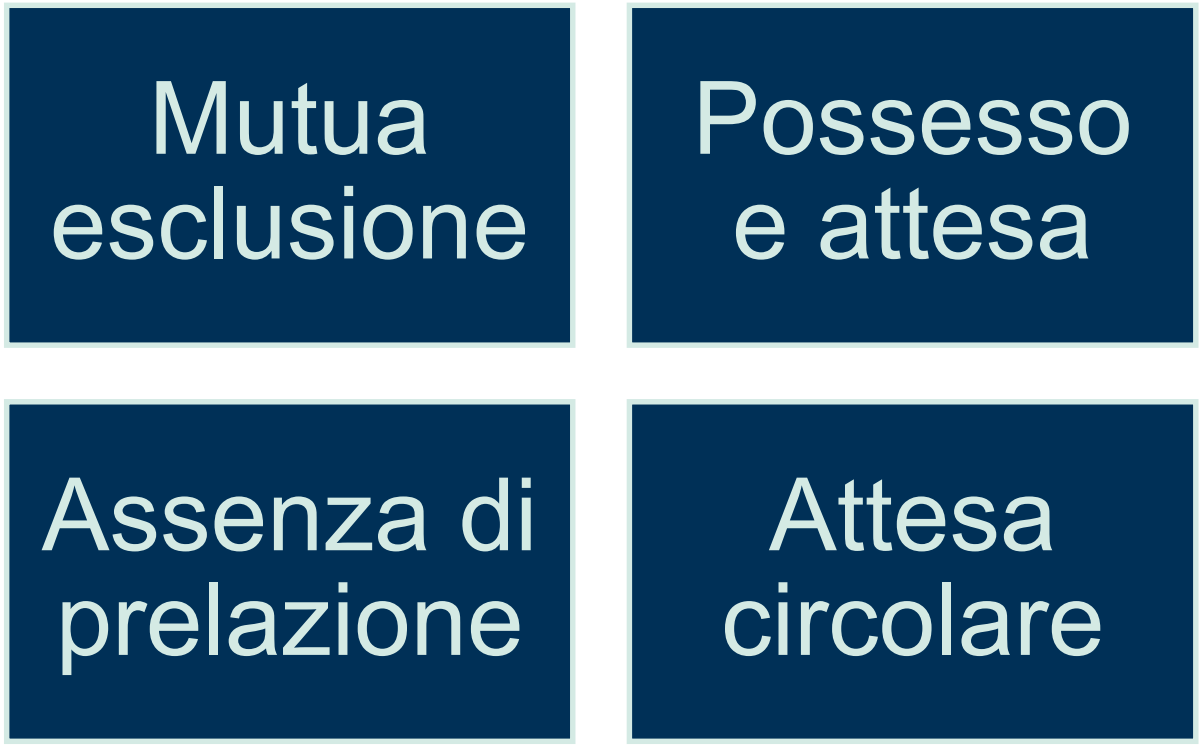
```
/* thread_one esegue in questa funzione */
void *do_work_one(void *param)
{
   int done = 0;

   while (!done) {
      pthread_mutex_lock(&first_mutex);
      if (pthread_mutex_trylock(&second_mutex)) {
         /**
          * Fa qualcosa
          */
         pthread_mutex_unlock(&second_mutex);
         pthread_mutex_unlock(&first_mutex);
         done = 1;
      }
      else
         pthread_mutex_unlock(&first_mutex);
   }

   pthread_exit(0);
}
```

```
/* thread_two esegue in questa funzione */
void *do_work_two(void *param)
{
   int done = 0;

   while (!done) {
      pthread_mutex_lock(&second_mutex);
      if (pthread_mutex_trylock(&first_mutex)) {
         /**
          * Fa qualcosa
          */
         pthread_mutex_unlock(&first_mutex);
         pthread_mutex_unlock(&second_mutex);
         done = 1;
      }
      else
         pthread_mutex_unlock(&second_mutex);
   }

   pthread_exit(0);
}
```

Figura 8.2 Esempio di stallo attivo.

Situazioni di stallo

Condizioni che generano una situazione di stallo



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**

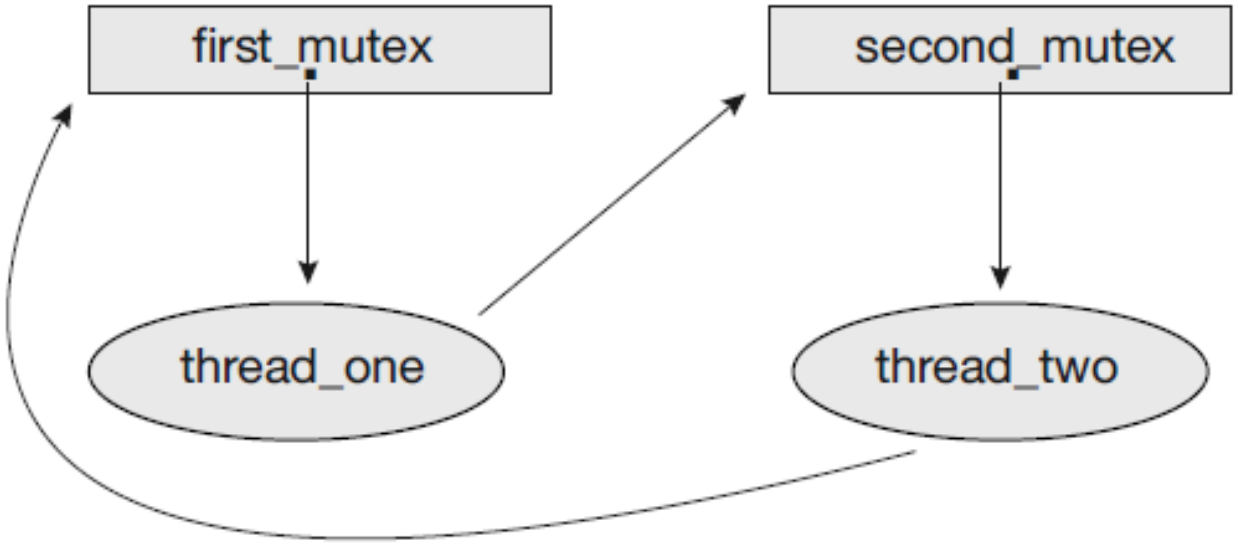


Figura 8.3 Grafo di assegnazione delle risorse per il programma nella Figura 8.1.

Esempio - Grafo di assegnazione delle risorse

- Insiemi T, R ed E:
  - $T = \{ T_1, T_2, T_3 \}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{ T_1 \rightarrow R_1, T_2 \rightarrow R_3, R_1 \rightarrow T_2, R_2 \rightarrow T_2, R_2 \rightarrow T_1, R_3 \rightarrow T_3 \}$
- Istanze delle risorse:
  - un'istanza del tipo di risorsa R_1
  - due istanze del tipo di risorsa R_2
  - un'istanza del tipo di risorsa R_3
  - tre istanze del tipo di risorsa R_4
- Stati dei thread:
  - il thread T_1 possiede un'istanza del tipo di risorsa R_2 e attende un'istanza del tipo di risorsa R_1
  - il thread T_2 possiede un'istanza dei tipi di risorsa R_1 ed R_2 e attende un'istanza del tipo di risorsa R_3
  - il thread T_3 possiede un'istanza del tipo di risorsa R_3

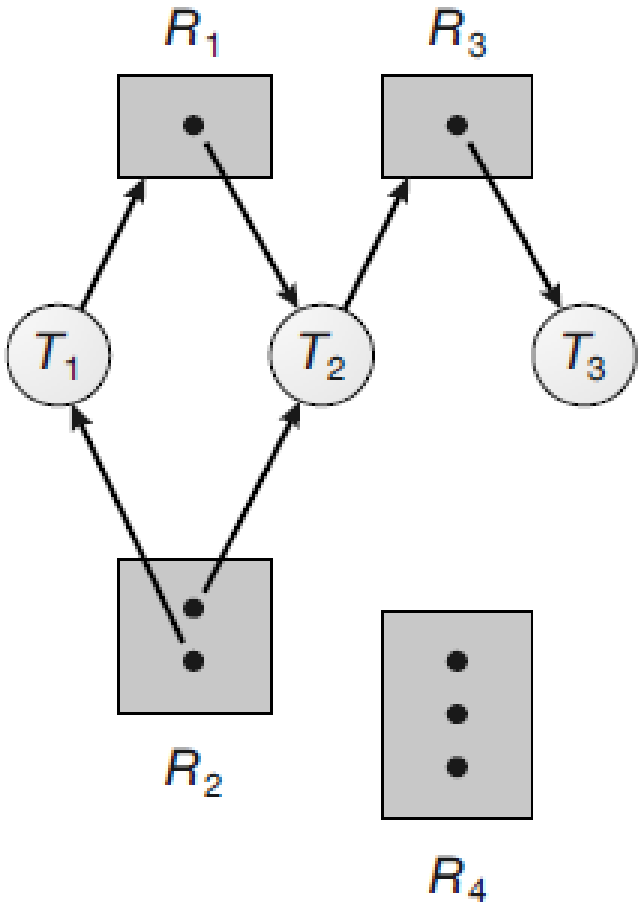


Figura 8.4 Grafo di assegnazione delle risorse.

Esempio - Grafo di assegnazione delle risorse con stallo

Se viene aggiunto un arco di richiesta $T_3 \rightarrow R_2$ al grafo della Figura 8.4 si viene a creare una situazione di stallo

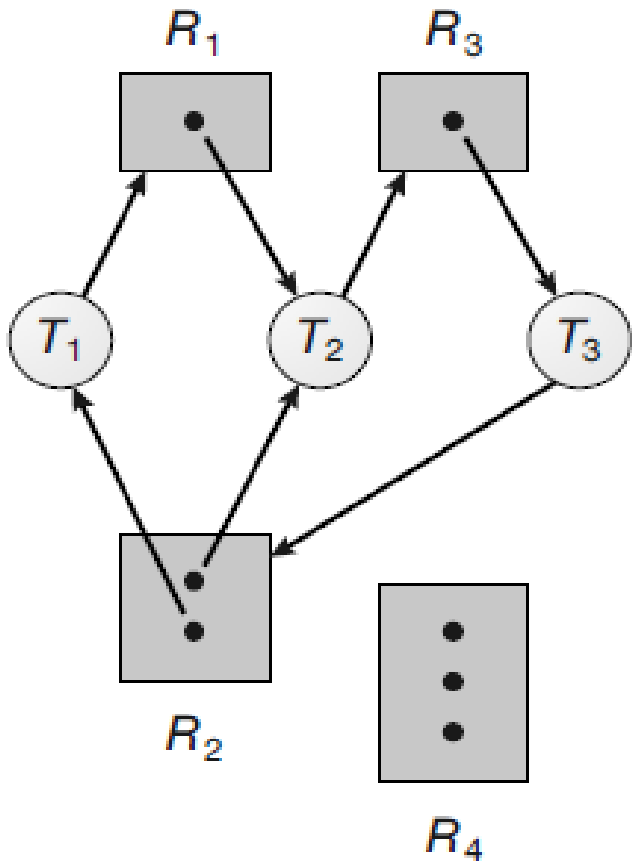


Figura 8.5 Grafo di assegnazione delle risorse con uno stallo.

Esempio - Grafo di assegnazione delle risorse con ciclo senza stallo

Anche in questo esempio c'è un ciclo:

$T_1 \rightarrow R_1 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$

In questo caso, però, non si ha alcuno stallo: il thread T_4 può rilasciare la propria istanza del tipo di risorsa R_2, che si può assegnare al thread T_3, rompendo il ciclo.

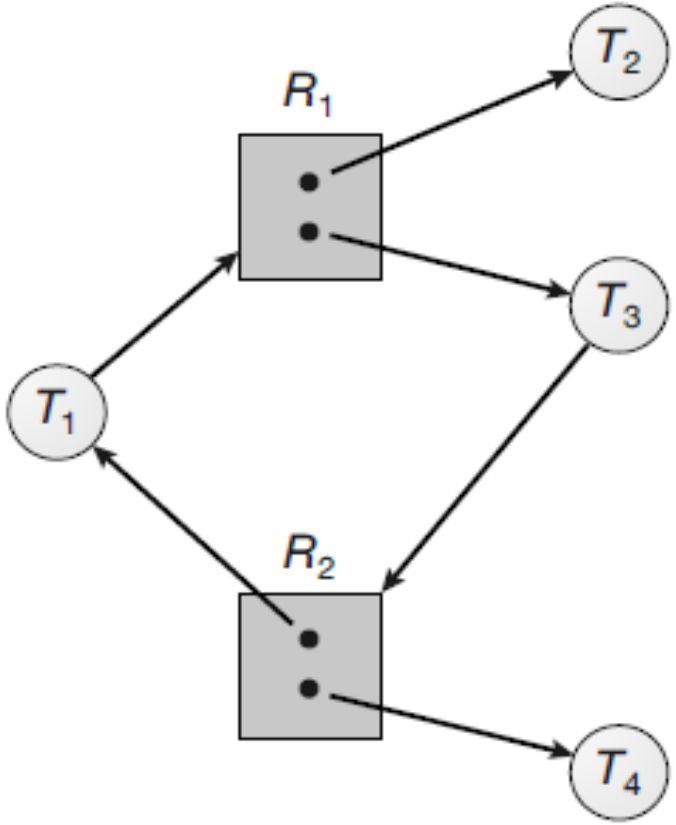


Figura 8.6 Grafo di assegnazione delle risorse con un ciclo, ma senza stallo.

Gestione delle situazioni di stallo

Il problema delle situazioni di stallo si può affrontare in tre modi:

1. ignorare del tutto il problema, *fingendo* che le situazioni di stallo non possano mai verificarsi nel sistema

2. usare un protocollo per prevenire o evitare le situazioni di stallo, assicurando che il sistema non entri *mai* in stallo

3. permettere al sistema di entrare in stallo, individuarlo e, quindi, eseguire il ripristino

Prevenire le situazioni di stallo

Prevenire le situazioni di stallo significa far uso di metodi atti ad assicurare che non si verifichi almeno una delle condizioni necessarie

- Mutua esclusione
- Possesso e attesa
- Assenza di prelazione
- Attesa circolare

Prevenire le situazioni di stallo

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.

Mutua esclusione → almeno una risorsa deve essere non condivisibile

Possesso e attesa → occorre garantire che un thread che richiede una risorsa non ne possegga altre

Assenza di prelazione → non deve essere possibile avere la prelazione su risorse già assegnate

Attesa circolare → imporre un ordinamento totale all'insieme di tutti i tipi di risorse e imporre che ciascun thread richieda le risorse in ordine crescente

Prevenire le situazioni di stallo

```
void transaction(Account from, Account to, double amount)
{
  mutex lock1, lock2;
  lock1 = get_lock(from);
  lock2 = get_lock(to);

  acquire(lock1);
    acquire(lock2);

      withdraw(from, amount);
      deposit(to, amount);

    release(lock2);
  release(lock1);
}
```

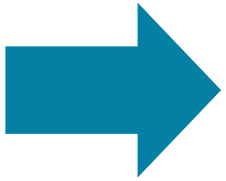
Figura 8.7 Esempio di stallo con ordinamento dei lock.

Evitare le situazioni di stallo

Per **evitare le situazioni di stallo** occorre che il sistema operativo abbia in anticipo informazioni aggiuntive riguardanti le risorse che un thread richiederà e userà durante le sue attività.

Evitare le situazioni di stallo

Il metodo per prevenire le situazioni di stallo illustrato in precedenza può causare effetti collaterali negativi



In alternativa: richiedere ulteriori informazioni sulle modalità di richiesta delle risorse

Evitare le situazioni di stallo

L'algoritmo per **evitare lo stallo** deve esaminare dinamicamente lo stato di assegnazione delle risorse per garantire che non possa esistere una condizione di attesa circolare

- Algoritmo con grafo di assegnazione delle risorse
- Algoritmo del banchiere
- Algoritmo di verifica della sicurezza
- Algoritmo di richiesta delle risorse

Stato sicuro

Uno **stato** si dice **sicuro** se il sistema è in grado di assegnare risorse a ciascun thread (fino al suo massimo) in un certo ordine e impedire il verificarsi di uno stallo.

Stato sicuro

Un sistema si trova in stato sicuro solo se esiste una **sequenza sicur**

Uno **stato sicuro** non è di stallo. Viceversa, uno stato di stallo è uno stato non sicuro; tuttavia *non* tutti gli stati non sicuri sono stati di stallo

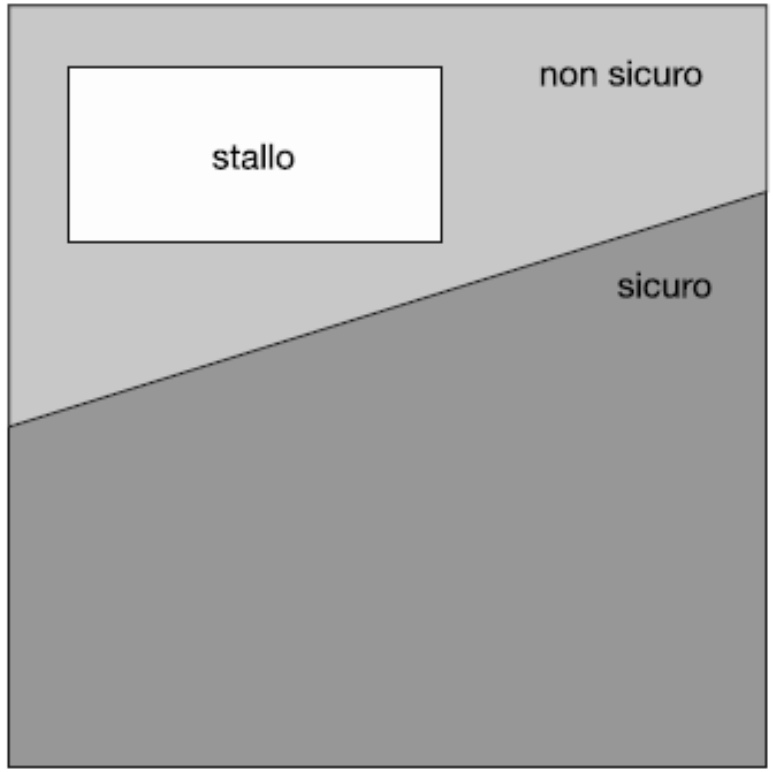
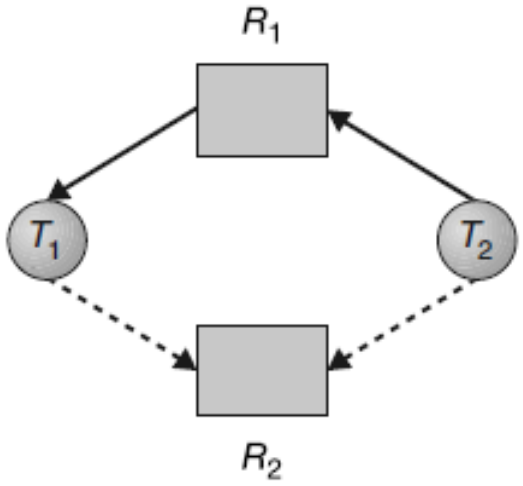
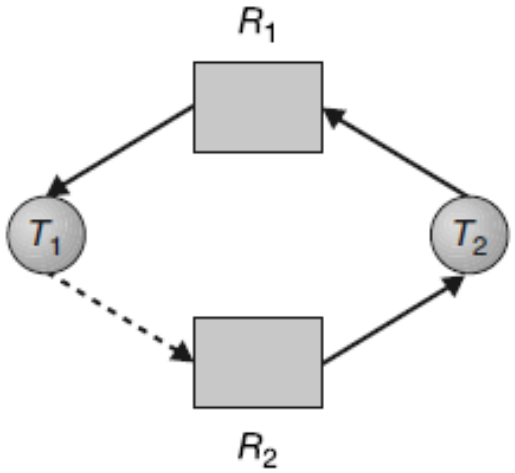


Figura 8.8 Spazi degli stati sicuri, non sicuri e di stallo.

Algoritmo con grafo di assegnazione delle risorse



- Un arco di rivendicazione $T_i \rightarrow R_j$ indica che un thread T_i *può richiedere la risorsa* R_j
- Viene indicato con una freccia tratteggiata



- Si supponga che T_2 *richieda* R_2
- Sebbene sia attualmente libera, R_2 *non* può essere assegnata a T_2, poiché quest'operazione creerebbe un ciclo nel grafo e un ciclo indica che il sistema è in uno stato non sicuro
- Se, a questo punto, T_1 *richiedesse* R_2, si avrebbe uno stallo

Algoritmo del banchiere

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, in modo da non poter più soddisfare le richieste di tutti i suoi clienti

Algoritmo del banchiere

La realizzazione dell'algoritmo del banchiere richiede la gestione di alcune strutture dati che codificano lo stato di assegnazione delle risorse del sistema.

Disponibili | *Massimo* | *Assegnate* | *Necessità*

Rilevamento delle situazioni di stallo

Istanza singola di ciascun tipo di risorsa

grafo d'attesa

variante del grafo di assegnazione delle risorse

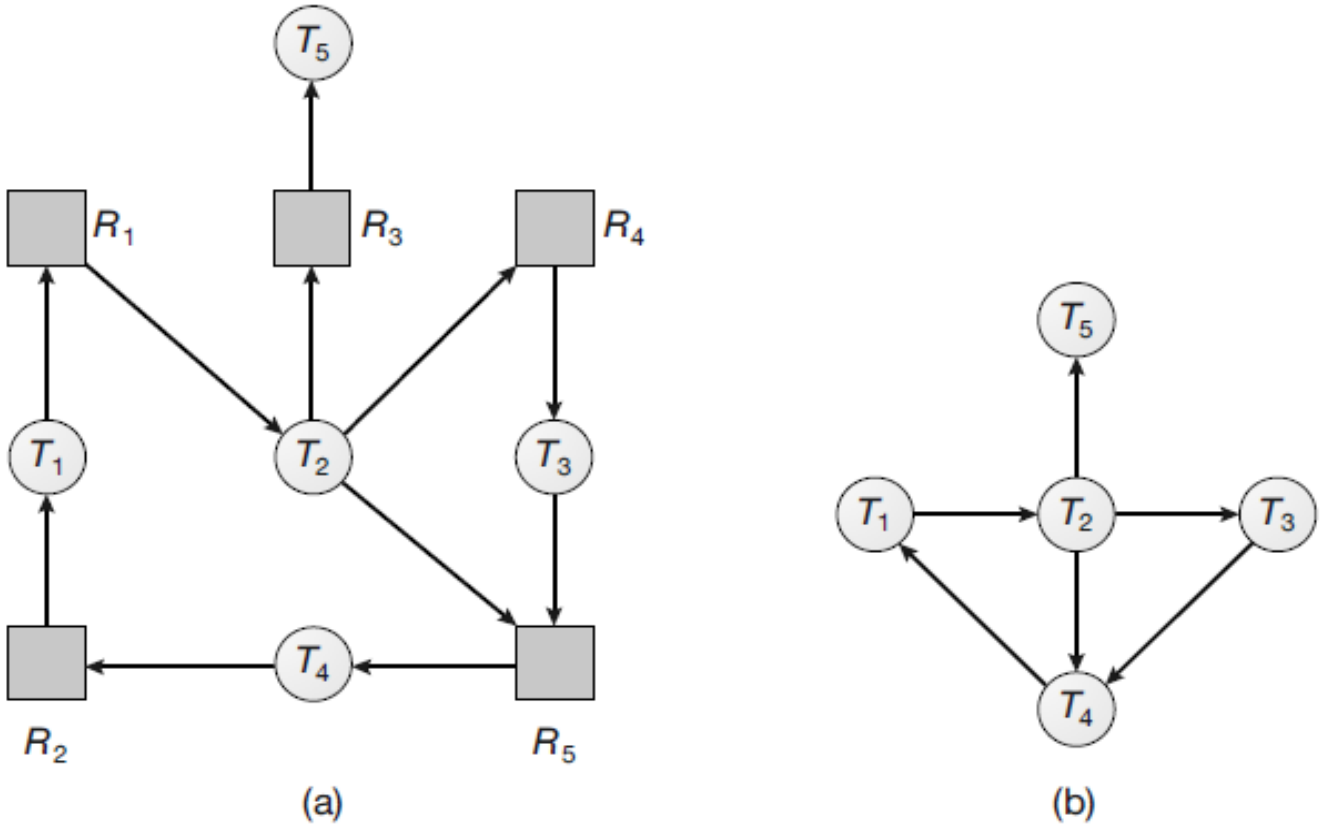


Figura 8.11 (a) Grafo di assegnazione delle risorse; (b) Grafo d'attesa corrispondente.

Rilevamento delle situazioni di stallo

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.

Rilevamento delle situazioni di stallo

Più istanze di ciascun tipo di risorsa

Esiste un algoritmo di rilevamento di situazioni di stallo che, invece, è applicabile a tali sistemi.
Esso si serve di strutture dati variabili nel tempo, simili a quelle adoperate nell'algoritmo del banchiere

Disponibili *Assegnate* *Richieste*

Rilevamento delle situazioni di stallo

Più istanze di ciascun tipo di risorsa

Uso dell'algoritmo di rilevamento

si ricorre all'algoritmo di rilevamento in base a

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

Ripristino da situazioni di stallo

Terminazione di processi e thread

Per eliminare le situazioni di stallo attraverso la terminazione di processi o thread si possono adoperare due metodi:

- Terminazione di tutti i processi in stallo
- Terminazione di un processo alla volta fino all'eliminazione del ciclo di stallo

Ripristino da situazioni di stallo

Prelazione delle risorse

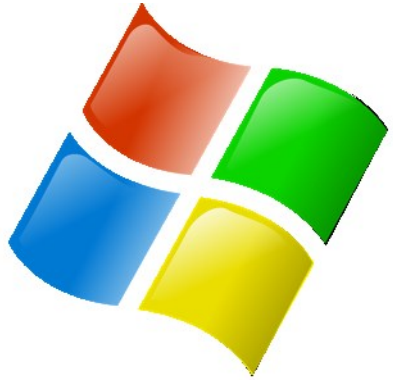
le risorse si sottraggono in successione ad alcuni processi e si assegnano ad altri finché si ottiene l'interruzione del ciclo di stallo.

Prelazione delle risorse

Ricorrendo alla prelazione delle risorse per l'eliminazione di uno stallo si devono considerare i seguenti problemi:

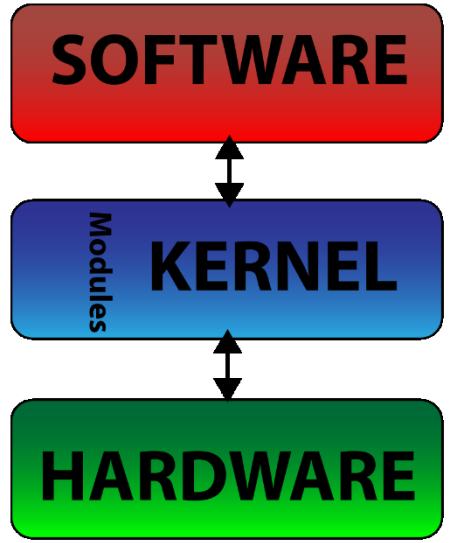
- Selezione di una "vittima"
- Ristabilimento di un precedente stato sicuro
- Attesa indefinita (starvation)

UNIVERSITÀ DEGLI STUDI DELLA BASILICATA



Corso di Sistemi Operativi
A.A. 2019/20

Stallo dei processi



Docente:
Domenico Daniele Bloisi

Novembre 2019