8. Stallo di Processi (deadlock)

- **Definizione**:

  situazione in cui ciascun processo in un insieme di n processi ($n \geq 2$) si trova in uno stato di *attesa* per il verificarsi di un evento che solo uno degli altri processi dell'insieme può provocare.

- **Risultato**:

  attesa infinita da parte di tutti gli n processi!

8. Stallo di Processi (deadlock)

- **Situazioni simili si verificano anche nella realtà**:

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone"

(da una vecchia norma di legge del Kansas)

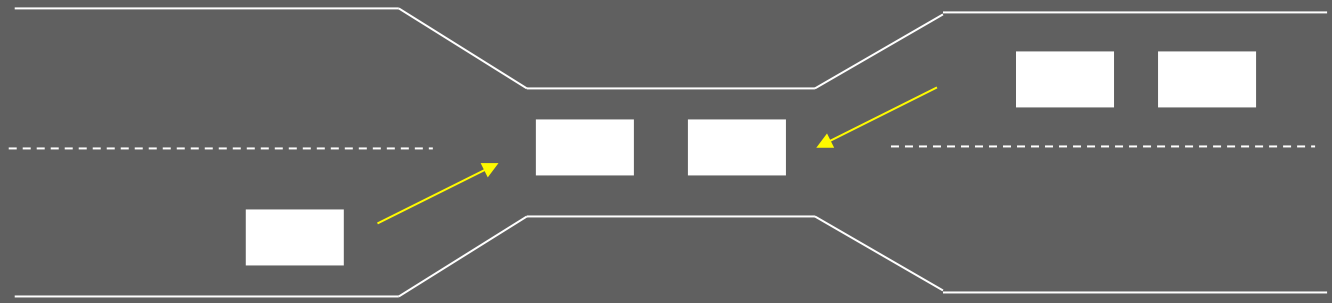
- I SO di oggi non affrontano il problema, ci devono pensare gli utenti.
- In futuro potrà diventare un compito dei SO

8. Stallo di Processi (deadlock)

I processi $\mathbf{P_1}$ e $\mathbf{P_2}$ usano entrambi uno dei due nastri e hanno bisogno di un secondo nastro. L'accesso ai nastri e' regolato dal semaforo *avail*, inizializzato a 2.

$\mathbf{P_1}$	$\mathbf{P_2}$
begin	**begin**
:	:
wait(avail)	wait(avail)
:	:
wait(avail)	wait (avail)
:	:
signal(avail)	signal(avail)
signal(avail)	signal (avail)
end	**end**

8. Stallo di Processi (deadlock)
Un ponte ad una sola corsia



- Ciascuna posizione di marcia può essere vista come una **risorsa**.
- Una situazione di deadlock può essere risolta se un auto **torna indietro** (libera una risorsa già occupata).
- Si verifica **starvation** se ciascuna auto sul ponte attende che l'altra liberi l'unica corsia di marcia

8.1 Modello del Sistema

- Un **sistema (HW + SO)** può essere visto come formato da:
- un insieme finito di **tipi di risorse R** (cicli di CPU, spazio di memoria, device di I/O*)*,
- Ogni tipo di risorsa è formata da un certo numero di istanze **indistinguibili** fra loro (ad esempio la ram può essere divisa in porzioni identiche, ciascuna delle quali può ospitare un processo)
- Un insieme di processi **P** che hanno bisogno di una o più istanze di alcune delle risorse per portare a termine la computazione

8.1 Modello del Sistema

- Si definisce **deadlock di un sottoinsieme** di processi del sistema $\{P_1, P_2, \ldots, P_n\} \subseteq \mathbf{P}$ la situazione in cui ciascuno degli n processi P_i e' in attesa del rilascio di una risorsa detenuta da uno degli altri processi del sottoinsieme; si forma cioè una **catena circolare** per cui:

$$P_1 \textit{ aspetta } P_2 \ldots \textit{ aspetta } P_n \textit{ aspetta } P_1$$

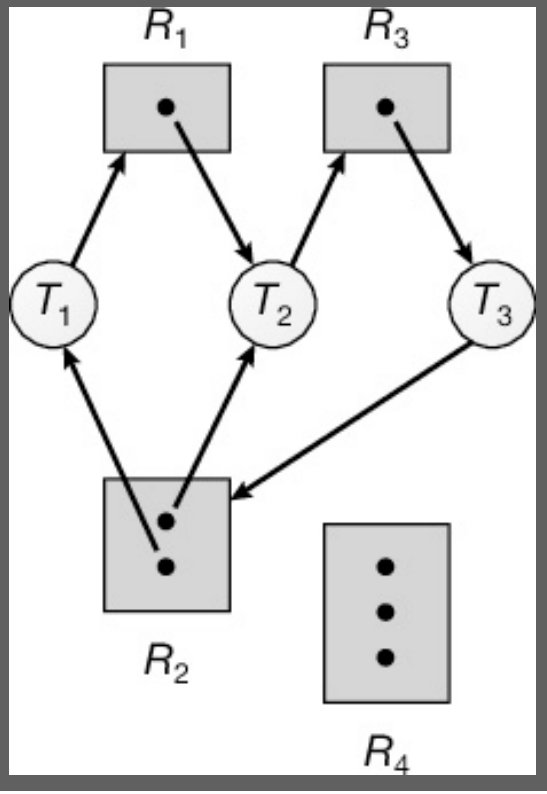
- Anche se non tutti i processi del sistema sono bloccati, la situazione non è desiderabile in quanto può bloccare alcune risorse, e dunque danneggiare anche i processi non coinvolti nel deadlock

8.3 Caratterizzazione del deadlock

- Il SO può avvalersi di una opportuna rappresentazione detta **grafo di assegnazione delle risorse** che in ogni istante registra quali risorse sono assegnate a quale processo, e quali risorse sta aspettando ciascun processo

- In questo esempio (fig. 8.5), c'è deadlock perché:
  T_1 attende che T_2 liberi R_1,
  T_2 attende che T_3 liberi R_3,
  T_3attende che T_1 o T_2 liberino R_2

- Ossia, il grafo contiene almeno un ciclo



8.3 Metodi per la gestione dei Deadlock

1. **Prevenire o evitare i deadlock**, usando un opportuno protocollo di richiesta e assegnamento delle risorse

2. lasciare che il deadlock si verifichi, ma fornire strumenti per la **scoperta e il recupero** dello stesso, esplorando il grafo di assegnazione delle risorse alla ricerca di cicli.

   Tuttavia, la soluzione 1 genera un eccessivo sottoutilizzo delle risorse, mentre la soluzione 2 non evita il problema e richiede lavoro al SO per eliminare il deadlock, dunque i SO moderni adottano la soluzione 3:

3. **lasciare agli utenti la prevenzione/gestione dei deadlock**