Sistemi Operativi

Corso di Laurea in Informatica

a.a. 2019-2020

SAPIENZA
UNIVERSITÀ DI ROMA

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

Esercitazione

Q1: Quante e quali modalità di esecuzione deve garantire la CPU (al minimo)?

Q1: Quante e quali modalità di esecuzione deve garantire la CPU (al minimo)?

R1: Almeno 2: kernel e user mode

Q2: Qual è la differenza tra kernel e user mode?

Q2: Qual è la differenza tra kernel e user mode?

R2: Quando la CPU si trova in kernel mode è in grado di eseguire istruzioni privilegiate; in user mode può eseguire solamente istruzioni non-privilegiate

Q3: La transizione da user a kernel mode avviene:

a. Quando un programma esegue una chiamata di funzione
b. Quando scade il quanto di tempo assegnato al processo in esecuzione
c. Quando un processo esegue una `fork()`
d. All'avvio del processo di bootstrap

Q3: La transizione da user a kernel mode avviene:

a. Quando un programma esegue una chiamata di funzione
b. Quando scade il quanto di tempo assegnato al processo in esecuzione
c. Quando un processo esegue una `fork()`
d. All'avvio del processo di bootstrap;

Q4: Le chiamate di sistema:

a. Consentono ad un programma di richiedere l'esecuzione di istruzioni privilegiate
b. Quando sono chiamate di I/O risultano sempre bloccanti
c. Devono essere implementate in spazio kernel
d. Causano la terminazione del processo in corso e l'avvio di un nuovo processo

Q4: Le chiamate di sistema:

a. Consentono ad un programma di richiedere l'esecuzione di istruzioni privilegiate
b. Quando sono chiamate di I/O risultano sempre bloccanti
c. Devono essere implementate in spazio kernel
d. Causano la terminazione del processo in corso e l'avvio di un nuovo processo

Q5: Qual è la differenza tra chiamate di sistema, eccezioni e interruzioni?

Q5: Qual è la differenza tra chiamate di sistema, eccezioni e interruzioni?

Chiamate di sistema → iniziate via software per ottenere un servizio dal sistema operativo (ad es., I/O)

Eccezioni → iniziate via software per rispondere ad una situazione anomala (ad es., divisione per 0)

Interruzioni → iniziate da dispositivi hardware per notificare un certo evento (ad es., timer)

Q6: Il System Call Handler:

a. Viene eseguito in spazio utente

b. Utilizza una tabella dedicata che contiene ogni chiamata di sistema supportata

c. Viene invocato periodicamente dallo scheduler della CPU

d. Salva e ripristina lo stato della computazione su appositi registri

Q6: Il System Call Handler:

a. Viene eseguito in spazio utente

b. Utilizza una tabella dedicata che contiene ogni chiamata di sistema supportata

c. Viene invocato periodicamente dallo scheduler della CPU

d. Salva e ripristina lo stato della computazione su appositi registri

Q7: Qual è la principale differenza tra programma e processo

Q7: Qual è la principale differenza tra programma e processo

R2: Un programma è un'entità statica rappresentata dal codice eseguibile ("testo"); un processo è un'entità dinamica il cui ciclo di vita è interamente gestito dal sistema operativo

Q8: Un processo che è in stato di attesa (wait) di un evento, una volta che questo si verifica:

a. Passa immediatamente nello stato running

b. Resta nello stato di attesa fino allo scadere del quanto di tempo

c. Ripristina il PCB ad esso associato

d. Nessuna delle precedenti

Q8: Un processo che è in stato di attesa (wait) di un evento, una volta che questo si verifica:

a. Passa immediatamente nello stato running

b. Resta nello stato di attesa fino allo scadere del quanto di tempo

c. Ripristina il PCB ad esso associato

d. Nessuna delle precedenti

Q8: Il Process Control Block (PCB)

a. È la struttura dati utilizzata dal sistema operativo per rappresentare ogni singolo processo
b. Contiene l'identificatore del processo e del relativo processo padre
c. Risiede in spazio utente
d. Contiene il valore del program counter e dello stack pointer

Q8: Il Process Control Block (PCB)

a. È la struttura dati utilizzata dal sistema operativo per rappresentare ogni singolo processo
b. Contiene l'identificatore del processo e del relativo processo padre
c. Risiede in spazio utente
d. Contiene il valore del program counter e dello stack pointer

Q9: Descrivere cosa si intende per context switch

Q9: Descrivere cosa si intende per context switch

R9: È la procedura con cui un processo attualmente in esecuzione sulla CPU, viene sostituito da un altro processo "schedulabile" (nella coda dei processi pronti). Per fare ciò è necessario che il sistema operativo salvi lo stato del processo corrente (nel suo PCB) e ripristini il PCB del processo da mandare in esecuzione

Q10: Un context switch scaturisce sempre a fronte di:

a. Una chiamata di sistema asincrona

b. Un segnale di interruzione da parte del timer

c. Una chiamata `fork()`

d. Passaggio in modalità kernel

Q10: Un context switch scaturisce sempre a fronte di:

a. Una chiamata di sistema asincrona

b. Un segnale di interruzione da parte del timer

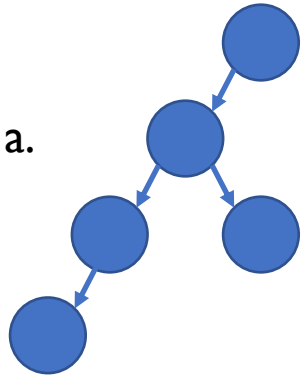
c. Una chiamata `fork()`

d. Passaggio in modalità kernel

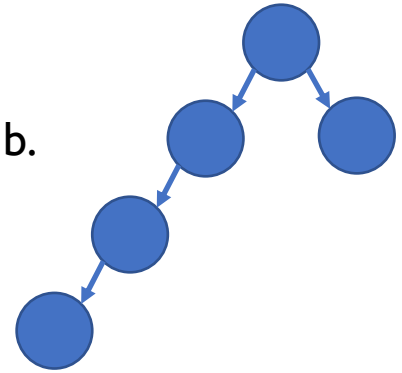
Q11: Indicare quale gerarchia di processi corrisponde al codice seguente:

```
int pid = fork();

if(pid == 0) {

    pid = fork();
    if (pid == 0) {
         …
    }
    else {
         pid = fork();
         if (pid == 0) {
            …
         }
         else {
            …
         }
    }
}
else {
    pid = fork();
    if(pid == 0) {
        …
    }
    else {
        …
    }
}
```

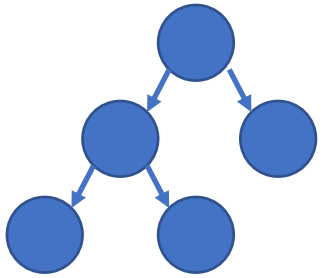
a.



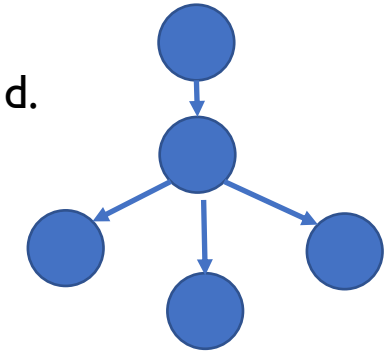
b.



c.



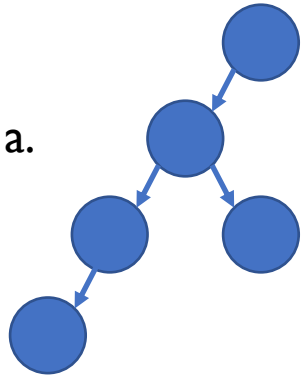
d.



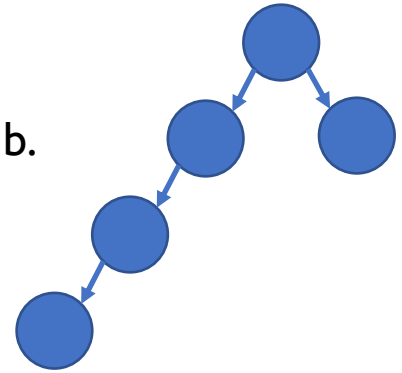
Q11: Indicare quale gerarchia di processi corrisponde al codice seguente:

```
int pid = fork();

if(pid == 0) { // A's first child (B)

    pid = fork();
    if (pid == 0) {// B's first child (C)
         …
    }
    else { // still in B
         pid = fork();
         if (pid == 0) { // B's second child (D)
            …
         }
         else { // still B
            …
         }
    }
}
else { // still in A
    pid = fork();
    if(pid == 0) { // A's second child (E)
        …
    }
    else {// still in A
        …
    }
}
```

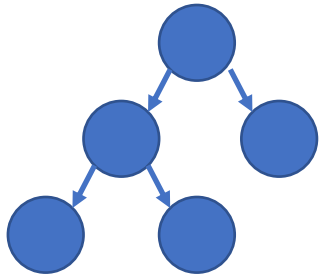
a.



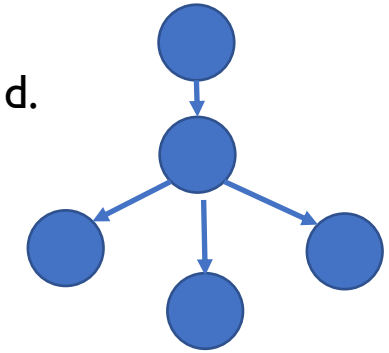
b.



c.



d.



Q12: In caso di sistemi *non-preemptive*, lo scheduler della CPU interviene:

a. Quando un nuovo processo viene creato o un processo esistente termina
b. Quando un processo passa dallo stato running a ready
c. Quando un processo passa dallo stato waiting a ready
d. Quando un processo passa dallo stato running a waiting

Q12: In caso di sistemi *non-preemptive*, lo scheduler della CPU interviene:

a. Quando un nuovo processo viene creato o un processo esistente termina
b. Quando un processo passa dallo stato running a ready
c. Quando un processo passa dallo stato waiting a ready
d. Quando un processo passa dallo stato running a waiting

Q13: Il tempo medio di attesa di un processo:

a. Dipende dal tempo di arrivo
b. Comprende il tempo speso dal processo bloccato in attesa di un evento (ad es., I/O)
c. È la metrica di riferimento da ottimizzare nei sistemi real-time (interattivi)
d. È minimo quando i processi arrivano in ordine di CPU-burst

Q13: Il tempo medio di attesa di un processo:

a. Dipende dal tempo di arrivo
b. Comprende il tempo speso dal processo bloccato in attesa di un evento (ad es., I/O)
c. È la metrica di riferimento da ottimizzare nei sistemi real-time (interattivi)
d. È minimo quando i processi arrivano in ordine di CPU-burst

Q14: Completare la tabella di seguito:

Job	$T_{arrival}$	T_{burst}	$T_{completion}$	$T_{turnaround}$	$T_{waiting}$
A	1	4	5		
B	2	1	6		
C	2	6	12		
D	3	5	17		
E	7	3	20		

Q14: Completare la tabella di seguito:

Job	$T_{arrival}$	T_{burst}	$T_{completion}$	$T_{turnaround}$	$T_{waiting}$
A	1	4	5	**4**	**0**
B	2	1	6	**4**	**3**
C	2	6	12	**10**	**4**
D	3	5	17	**14**	**9**
E	7	3	20	**13**	**10**

Q15: Calcolare il tempo medio d'attesa (average waiting time) dei seguenti processi, assumendo una politica di scheduling *round robin* con time slice = 3, nessuna attività di I/O e context switch trascurabile

Job	$T_{arrival}$	T_{burst}
A	0	5
B	3	8
C	4	2

Q15: Calcolare il tempo medio d'attesa (average waiting time) dei seguenti processi, assumendo una politica di scheduling *round robin* con time slice = 3, nessuna attività di I/O e context switch trascurabile

Job	$T_{arrival}$	T_{burst}
A	0	5
B	3	8
C	4	2

avg. waiting time = (5 + 4 + 2)/3 ~ **3.7**

Q16: Completare la tabella di seguito, assumendo che il quanto di tempo sia pari a 2, non vi sia I/O, e che tutti i processi arrivino al tempo t=0

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	80				
B	20				
C	50				
D	30				
E	20				
Avg.					

Q16: Completare la tabella di seguito, assumendo che il quanto di tempo sia pari a 2, non vi sia I/O, e che tutti i processi arrivino al tempo t=0

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	80	80		0	
B	20	100		80	
C	50	150		100	
D	30	180		150	
E	20	200		180	
Avg.		**142**		**102**	

Q16: Completare la tabella di seguito, assumendo che il quanto di tempo sia pari a 2, non vi sia I/O, e che tutti i processi arrivino al tempo t=0

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	80	80	200	0	120
B	20	100	94	80	74
C	50	150	170	100	120
D	30	180	130	150	100
E	20	200	100	180	80
Avg.		**142**	**~139**	**102**	**~99**

Q17: Lo scheduling Shortest Job First (SJF):

a. Funziona solo in combinazione con sistemi non-preemptive

b. Privilegia implicitamente i processi I/O-bound

c. Privilegia implicitamente i processi CPU-bound

d. È un esempio di priority scheduling

Q17: Lo scheduling Shortest Job First (SJF):

a. Funziona solo in combinazione con sistemi non-preemptive

b. Privilegia implicitamente i processi I/O-bound

c. Privilegia implicitamente i processi CPU-bound

d. È un esempio di priority scheduling

Q18: È preferibile progettare un'applicazione multi-thread anziché multi-process perché:

a. I context switch tra thread sono meno onerosi

b. È possibile ottenere parallelismo reale in caso di architetture multi-processor

c. La comunicazione intra-thread è più veloce

d. Non necessita di chiamate di sistema

Q18: È preferibile progettare un'applicazione multi-thread anziché multi-process perché:

a. I context switch tra thread sono meno onerosi
b. È possibile ottenere parallelismo reale in caso di architetture multi-processor
c. La comunicazione intra-thread è più veloce
d. Non necessita di chiamate di sistema

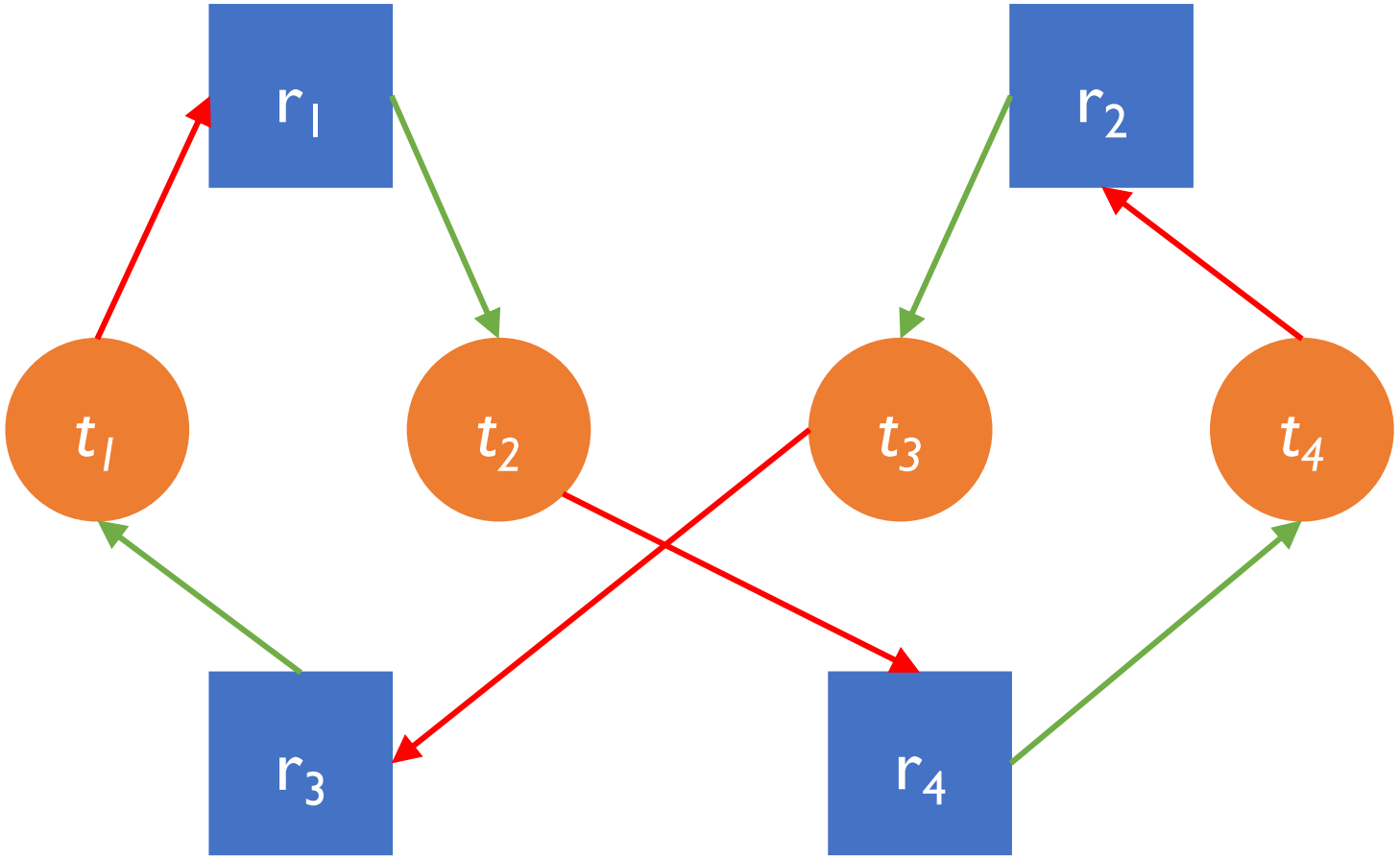
Q19: Descrivere le controindicazioni di un'implementazione user thread "pura"

Q19: Descrivere le controindicazioni di un'implementazione user thread "pura"

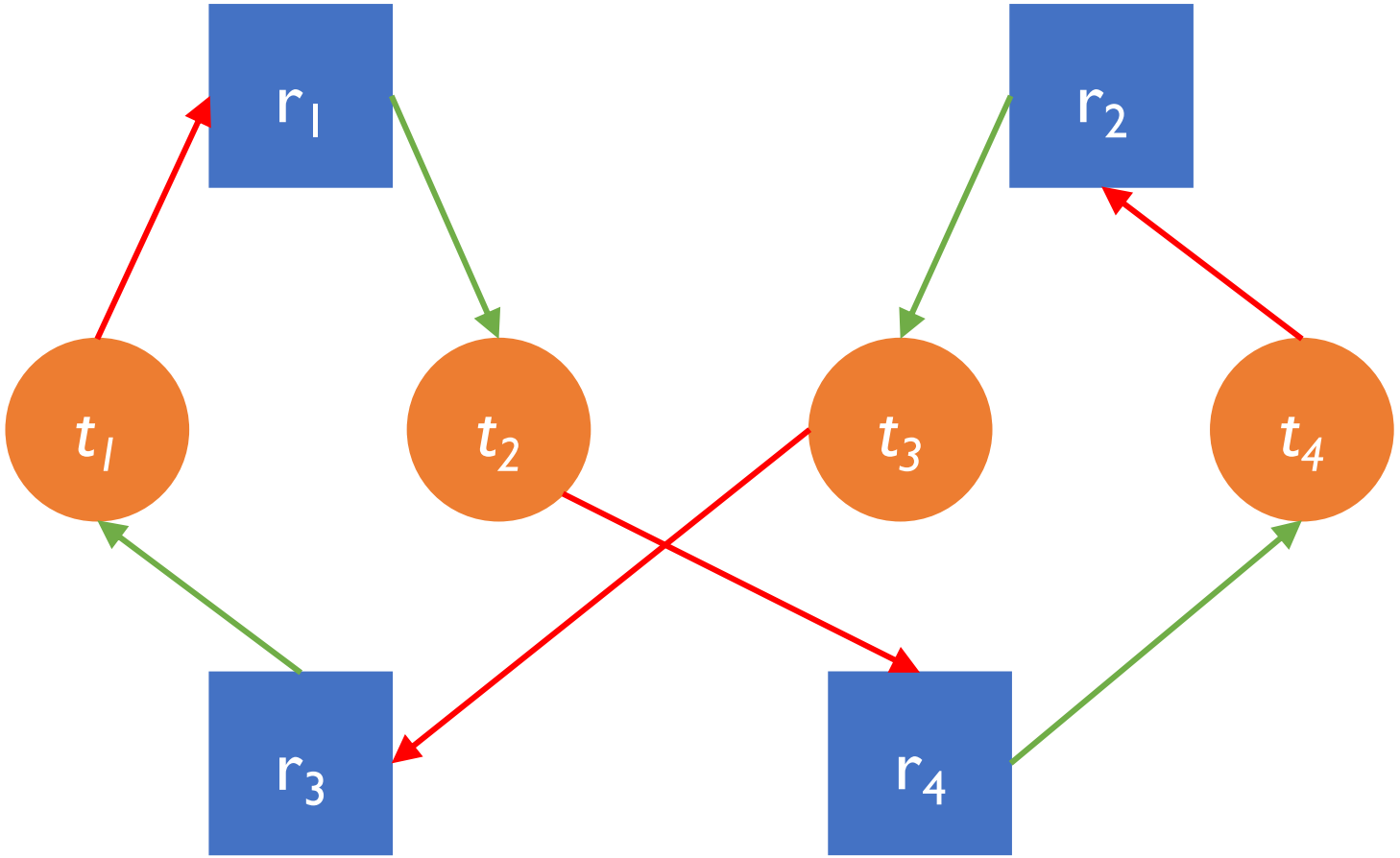
R19:

- Nessuna concorrenza/parallelismo reale
- Le politiche di scheduling possono essere non ottimali
- Occorre prevedere chiamate di sistema non bloccanti, altrimenti tutti i thread all'interno dello stesso processo saranno bloccati

Q20a: Dato il seguente RAG, indicare se si è in presenza di possibile deadlock

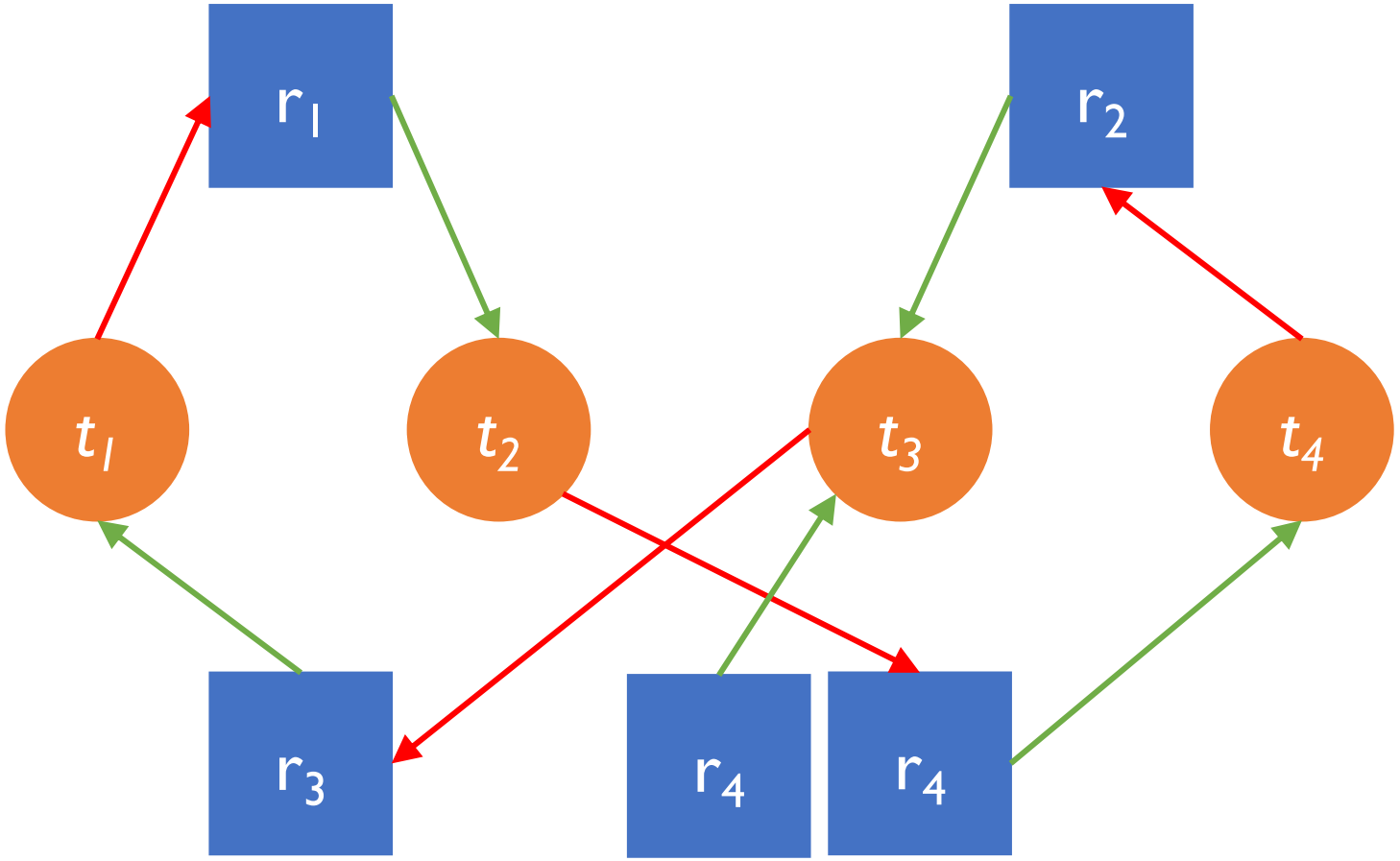


Q20a: Dato il seguente RAG, indicare se si è in presenza di possibile deadlock

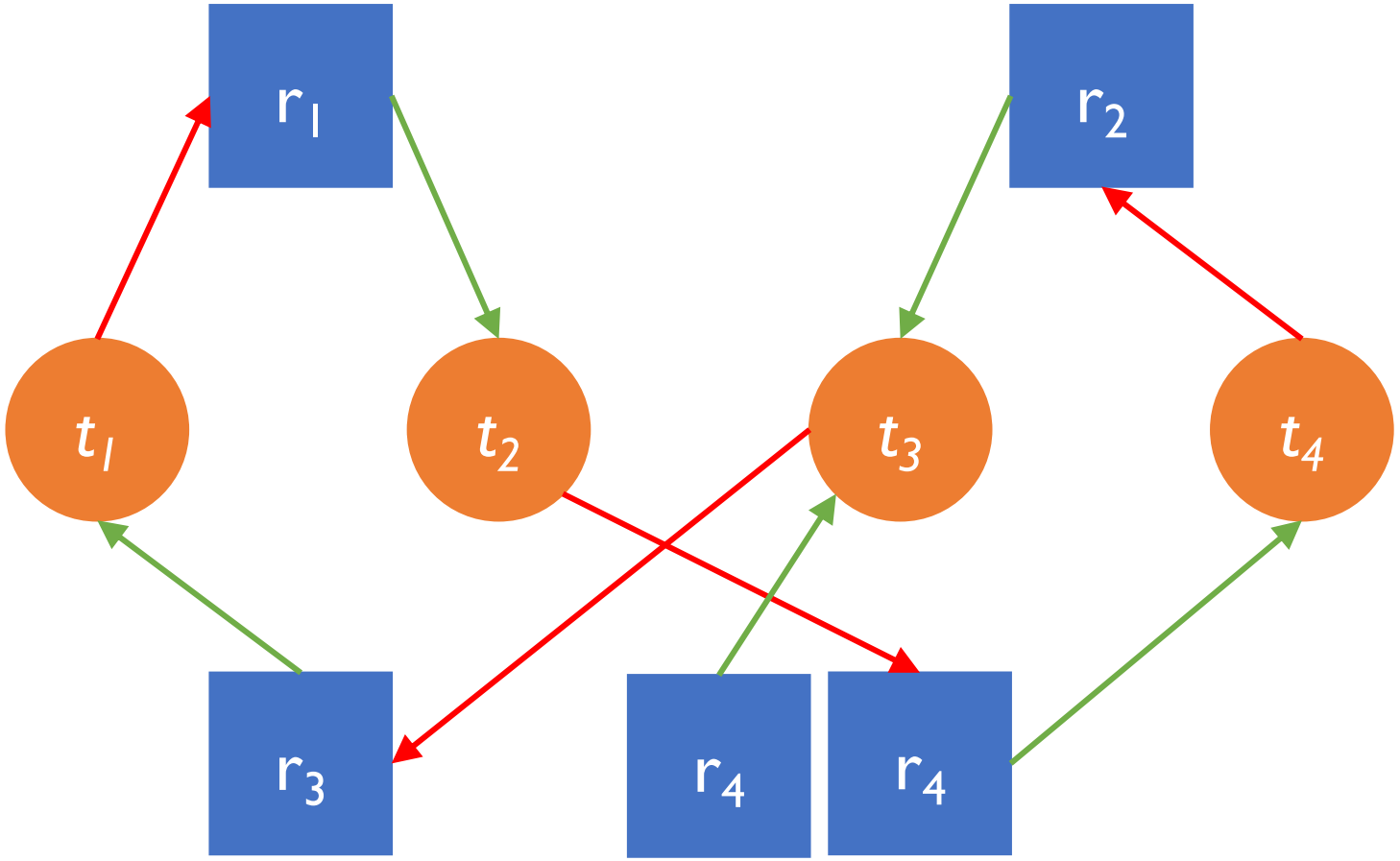


Sì, il RAG contiene un ciclo

Q20b: L'aggiunta di una risorsa r_4 risolverebbe il problema?

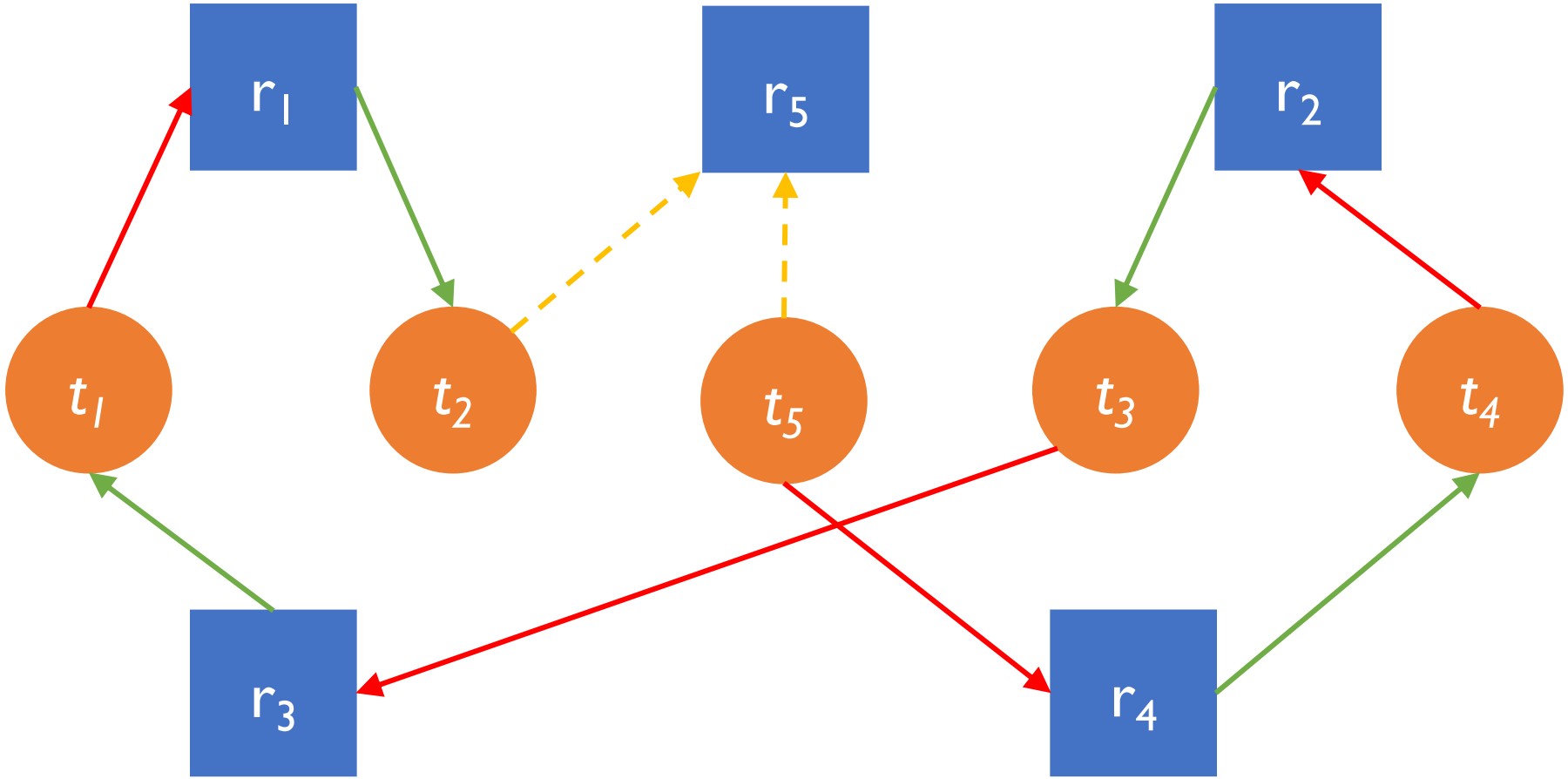


Q20b: L'aggiunta di una risorsa r_4 risolverebbe il problema?

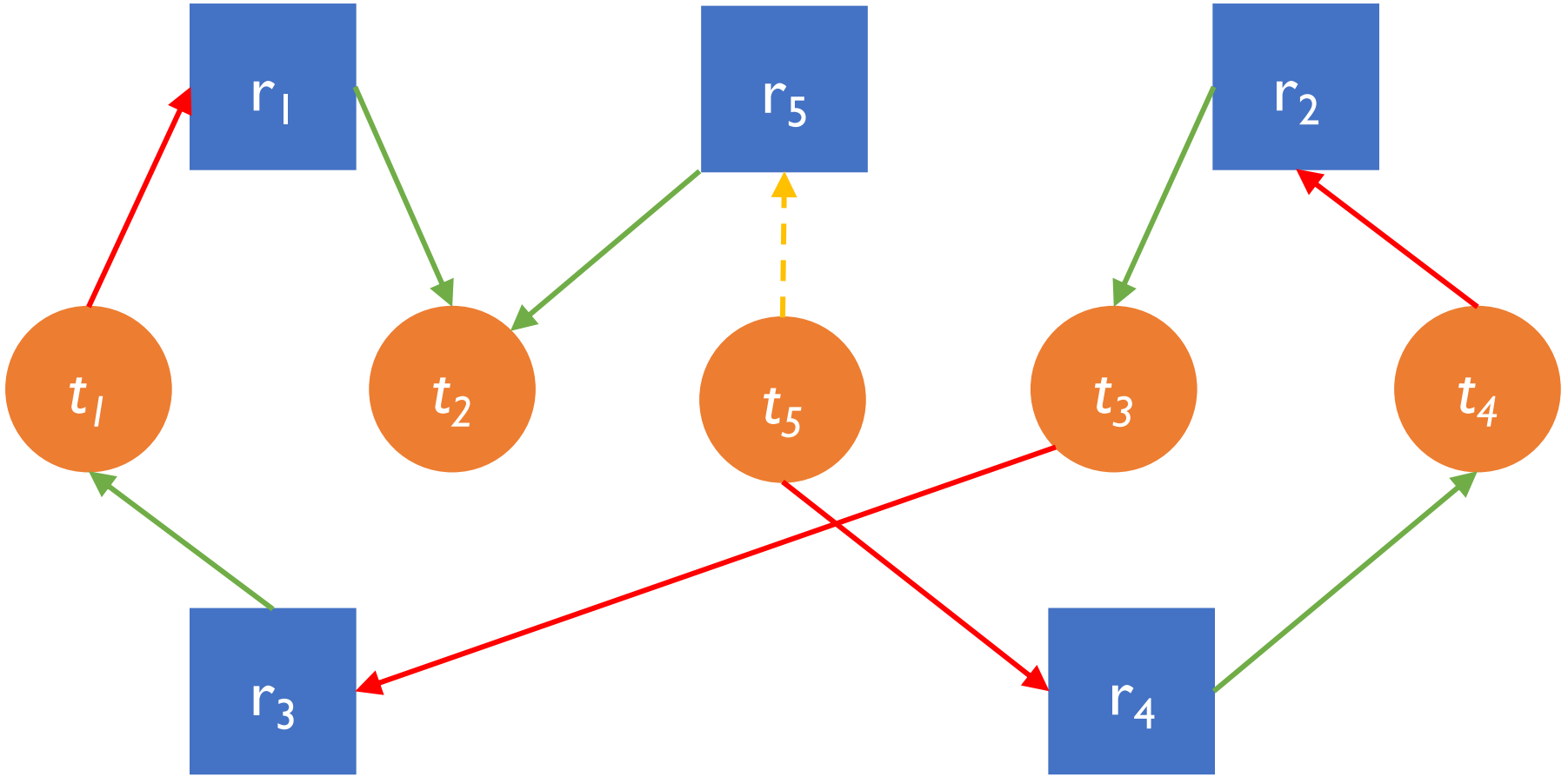


No, poiché la risorsa extra r_4 è assegnata a t_3 che fa anch'esso parte del ciclo

Q20c: Dato il seguente RAG, a quale thread il sistema assegnerà la risorsa r_5?



Q20c: Dato il seguente RAG, a quale thread il sistema assegnerà la risorsa r_5?



t_2 per evitare di transire in uno stato unsafe