UNIVERSITÀ DEGLI STUDI DI VERONA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Eserciziario di Sistemi Operativi Avanzati

Docente: dott. Graziano Pravadelli

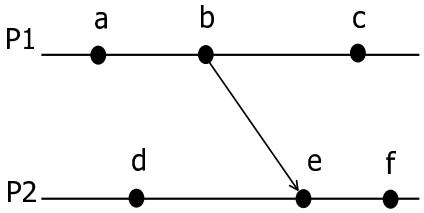
A cura di: Gabriele Pozzani e Stefano Pigozzo

Università di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
19 Giugno 2002

---

1. Descrivere le caratteristiche principali e mostrare le principali differenze tra S.O. basati su microkernel ed exokernel **[4 punti]**

2. Si considerino due processi P_1 e P_2, in esecuzione su due nodi distinti, sincronizzati con un sistema di *clock logici*, e la sequenza di eventi rappresentata in figura.



   Completare il diagramma aggiungendo il relativo timestamp $C()$ ad ogni evento.

   Si individui poi una coppia di eventi e_1 ed e_2 ($e_1, e_2 \in \{a...f\}$) tali per cui:

   - I relativi timestamp $C(e_1)$ e $C(e_2)$ stiano nella relazione $C(e_1) < C(e_2)$;
   - $e_1 \not\rightarrow e_2$

   Mostrare quindi un diagramma temporale alternativo basato su *vettori di clock logici* che risolva l'anomalia precedente, cioè tale per cui, se $e_1 \not\rightarrow e_2$, allora $C(e_1) \not< C(e_2)$ **[7 punti]**

3. Descrivere le caratteristiche principali di NFS, mostrandone sia uno schema architetturale sia uno schema funzionale. **[5 punti]**

4. Descrivere il funzionamento del meccanismo di inversione di priorità, e mostrare un esempio di come esso può portare a situazioni di deadlock. **[6 punti]**

5. Si consideri il seguente insieme di taski S:

Task	T_i	C_i
τ_1	20	2
τ_2	15	4
τ_3	10	3
τ_4	20	5

   Si mostri il diagramma temporale relativo allo scheduling di S secondo i seguenti algoritmi:

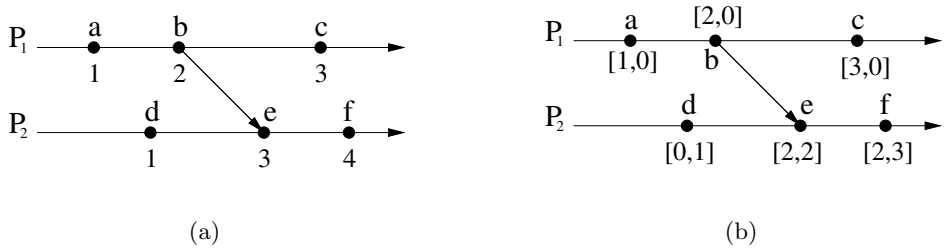
   - (Rate Monotonic) RM
   - EDF

   Confrontarli in termini dei tempi medi di risposta e della massima *lateness*. **[5 punti]**

SOLUZIONE:

1) Nei μkernel, come nei kernel, si hanno solo due livelli: livello kernel e livello utente.
Il μkernel include solo i servizi essenziali mentre tutti i rimanenti servizi sono forniti da processi esterni al kernel che vengono implementati come processi server. La comunicazione con tali processi avviene sempre tramite il kernel che quindi svolge la funzione di intermediario.
Nell'exokernel le idee e le caratteristiche del μkernel sono portate all'estremo. Infatti l'exokernel fornisce soltanto il multiplexing dell'hardware tra i diversi processi. Tutte le altre astrazioni vengono invece fornite da librerie a livello utente dette LOS (Library Operating System). Dal punto di vista dell'hardware si ha quindi la separazione tra protezione, fornita dal kernel, e gestione che invece è a carico delle singole applicazioni.

2) Utilizzando per entrambi i nodi un valore d=1 ed applicando l'algoritmo di Lamport si ottiene quanto mostrato nella figura 1(a).



(a) (b)

Figura 1:

Ma tale soluzione viola la causalità dato che se prendiamo la coppia di eventi (d,c) si ha che C(d)<C(c) ma d$\nrightarrow$c. Utilizzando invece i vettori di clock si ottiene quanto mostrato nella figura 1(b) che risolve il problema della violazione della causalità infatti osservando ancora la coppia di eventi (d,c) ora si ha che C(d)$\not<$C(c) e pi in generale si può dire che C(e_1)<C(e_2) sse $e_1 \rightarrow e_2$.

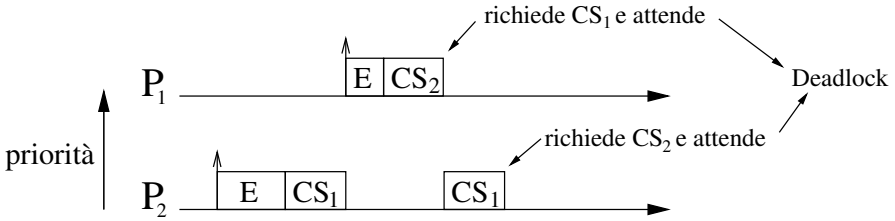
3) NFS è basato su un'architettura client/server, anche se la distinzione tra i due è puramente concettuale dato che essi eseguono lo stesso software.
Si chiama server un host che mette a disposizione (esporta)una o pi parti del prorpio file system. I file system epsortati devono essere registrati nel file /etc/exports. Il server richiede in esecuzione 3 demoni:

- nfsd che accetta le richieste dai client e le passa a mountd;
- mountd che gestisce le richieste;
- rpcbind che consente ai client di conoscere la porta su cui è in attesa il server NFS.

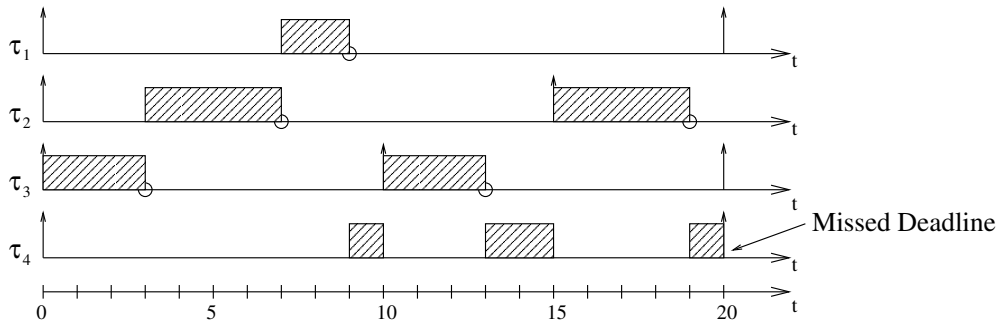
I client possono integrare nel proprio file system locale i file system del server eseguendone il mount in un mount point locale. È inoltre possibile eseguire l'automounting all'avvio del client registrando opportunamente i file system nel file /etc/fstab.
Il protocollo di mount prevede che il client invii una richiesta al server contenente il path del file system a cui è interessato. Il server verifica che il path sia esportato e in caso affermativo ritorna al client un file handle che verrà utilizzato dal client per tutte le operazioni successive.
NFS prevede anche un protocollo per l'accesso a file contenente delle primitive per tutte le operazioni che possono essere eseguite sui file.
NFS è basato su RPC e di conseguenza è stateless ed indipendente dal protocollo di trasporto utilizzato dalla rete sottostante.
L'architettura NFS è divisibile in tre strati:

1. system call layer;
2. Virtual File System (VFS) layer: mantiene una tabella di v-node, uno per ogni file aperto. Ogni v-node punta ad un i-node se il file è locale o ad un r-node se è remoto;
3. NFS layer: traduce le richieste secondo il protocollo NFS.

4) Si ha inversione di priorità quando un processo ad alta priorità è costretto a sospendere la propria esecuzione a tempo indefinito in attesa che un proceso a pi bassa priorità liberi una sezione critica in comune ad entrambi.
Il caso pi comune si ha quando P_2 entra in una sezione critica, viene sospeso per poter eseguire P_1 ($Prio(P_1)>Prio(P_2)$) ma questo a sua volta si blocca perché ha bisogno di entrare nella sezione critica ancora occupata da P_2. A questo punto P_2 torna in esecuzione e P_1 deve attendere. Il problema maggiore risulta però dal fatto che P_2 a sua volta può ora essere sospeso per eseguire altri processi a pi alta priorità aumentando così il tempo per cui P_1 rimane bloccato.
L'inversione di priorità può anche portare a deadlock. Consideriamo infatti il caso in cui P_1 (ad alta priorità necessiti di due sezioni critiche: CS_2 e CS_1 (innestata in CS_2), mentre P_2 (a bassa priorità necessiti di CS_1 e CS_2 (innestata in CS_1). Allora è possibile che i due processi vadano in deadlock come mostrato nella figura sottostante.

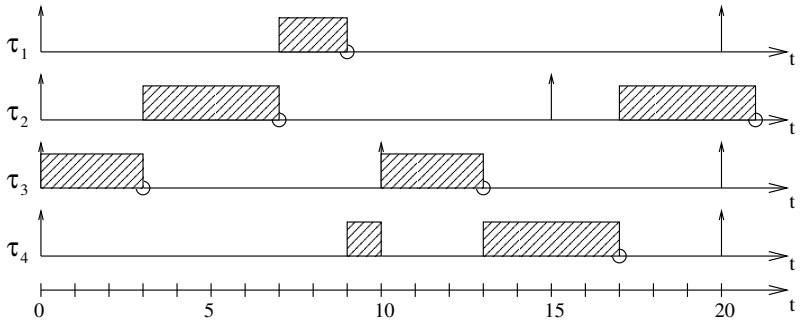


5) a. In RM la priorità viene fissata in modo inversamente proporzionale al periodo dei task. Si ha quindi che $p(\tau_3) > p(\tau_2) > p(\tau_1) > p(\tau_4)$. Allora si ottiene il seguente scheduling:



Si vede come RM non riesca a schedulare correttamente l'insieme dei task.

b. Utilizzando EDF e considerando che a parità di deadline viene scelto il processo con l'indice minore s ottiene il seguente diagramma temporale:



Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
10 Luglio 2002

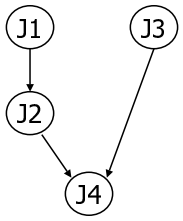
---

1. Descivere il funzionamento dell'algoritmo di Ricart & Agrawala per la gestione distribuita della sincronizzazione tra un insieme di processi, mostrando anche un esempio di funzionamento.

   Si evidenzino inoltre vantaggi e svantaggi rispetto ad un approccio centralizzato. **[5 punti]**

2. Discutere il problema dell'inversione di priorità, e mostrare una possibile soluzione tramite il metodo del *priority ceiling* **[5 punti]**

3. Sia dato il seguente insieme di task aperiodici, legati dal seguente grafo delle dipendenze:



Task	C_i	d_i
T_1	3	5
T_2	2	6
T_3	1	3
T_4	3	9

   Fornire una schedulazione valida dei task utilizzando l'algoritmo LDF. **[5 punti]**

4. Costruire un insieme di tre task periodici che sia schedulabile, e per il quale il test di utilizzazione U fallisca.

   NOTA: $U(3) \approx 0.78$. **[6 punti]**

5. Si consideri il seguente insieme di task periodici, ed il seguente insieme di richieste (task) aperiodiche:

Task	a_i	C_i	T_i
τ_1	0	2	6
τ_2	0	1	4

Task	a_i	C_i	d_i
J_1	5	2	11
J_2	10	1	16

   dove le deadline si intendono come **assolute**. Si assuma che i task periodici vengano schedulati secondo un algoritmo RM (rate monotonic).

   Assumendo che la gestione delle richieste aperiodiche avvenga tramite un *priority server* di tipo *polling*, a cui sia assegnata una capacità $C_s = 2$, determinare il valore **massimo** del periodo del server che permetta la schedulazione dell'insieme dei task. **[9 punti]**

SOLUZIONE:

1) L'algoritmo di Ricart & Agrawala funziona come segue:

- quando un processo P_i desidera entrare nella sezione critica genera un timestamp TS ed invia in modo affidabile (con acknowledgement) un messaggio di *request*(P_i, CS_k, TS) a tutti gli altri processi del sistema.
- quando P_j riceve un messaggio di richiesta si possono verificare 3 casi:
  1. se P_j è nella SC accoda il messaggio di *reply* a P_i;
  2. se P_j non è nella SC e non vuole nemmeno entrarvi invia immediatamente un messaggio di *reply* a P_i;
  3. se P_j non è nella SC ma vuole entrarvi allora confronta il timestamp TS_j della propria richiesta con TS, se TS_j >TS allora invia il messaggio di *reply* a P_i altrimenti accoda il messaggio di *reply* per P_i.
- quando P_i riceve un messaggio di *reply* da tutti gli altri processi può entrare nella SC.
- quando P_i esce dalla SC invia un messaggio di *reply* a tutte le richieste accodate nel frattempo.

Un'esempio di funzionamento è il seguente:

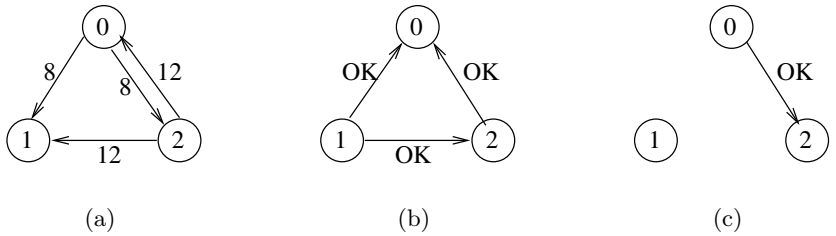


Figura 2:

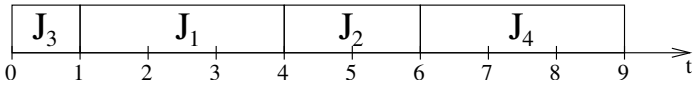
P_0 e P_2 richiedono l'accesso alla stessa risorsa rispettivamente agli istanti 8 e 12, entrambi quindi inviano un messaggio di richiesta a tutti gli altri processi del sistema indicando il proprio timestamp (Figura 2(a)). P_1 non essendo interessato alla risorsa invia immediatamente i messaggi di reply a P_0 e P_2, lo stesso fa P_2 dato che il proprio timestamp è maggiore di quello di P_0, mentre infine P_0 accoda il messaggio di reply per P_2 ed accede alla risorsa (Figura 2(b)). Quando P_0 esce dalla sezione critica invia il messaggio di reply a P_2 che può così accedere alla risorsa condivisa (Figura 2(c)).

2) Si ha inversione di priorità quando un processo ad alta priorità è costretto a sospendere la propria esecuzione a tempo indefinito in attesa che un proceso a pi bassa priorità liberi una sezione critica in comune ad entrambi.
Il caso pi comune si ha quando P_2 entra in una sezione critica, viene sospeso per poter eseguire P_1 (Prio(P_1)>Prio(P_2)) ma questo a sua volta si blocca perché ha bisogno di entrare nella sezione critica ancora occupata da P_2. A questo punto P_2 torna in esecuzione e P_1 deve attendere. Il problema maggiore risulta però dal fatto che P_2 a sua volta può ora essere sospeso per eseguire altri processi a pi alta priorità aumentando così il tempo per cui P_1 rimane bloccato.
Un meccanismo per la gestione dell'inversione di priorità è il *priority ceiling* che funziona come segue:

- si associa ad ogni risorsa S_k una priorità C(S_k) uguale alla priorità del processo a pi alta priorità che può accedere alla risorsa stessa.
- sia J_i il processo a pi alta priorità pronto per l'esecuzione, allora J_i viene assegnato alla CPU.
- sia S^* la risorsa con la pi alta priorità tra tutte le risorse già bloccate da un qualunque processo $J_n \neq J_i$.
- se J_i richiede la risorsa s_k la può ottenere se e solo se P(J_i)>C(S^*) altrimenti viene bloccato su S^* e non può acquisire S_k.
- quando J_i viene bloccato su una risorsa esso trasmette la propria priorità al processo J_k che possiede quella risorsa. Quindi J_k torna in esecuzione con priorità P(J_i).

- quando J_k esce alla sezione critica, rilascia la risorsa e sveglia il processo api alta priorità che era bloccato su quella risorsa. Inoltre la priorità di J_k viene impostata uguale alla priorità del processo a pi alta priorità ancora bloccato sulla risorsa, se invece non sono presenti altri processi bloccati J_k torna ad avere la sua priorità originale.
- N.B.: la trasmissione della priorità è transitiva.

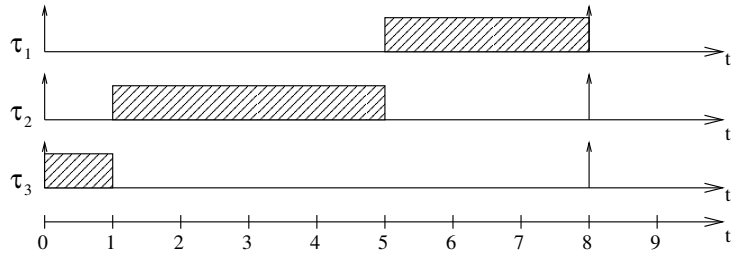
3) Ricordando che LDF costruisce lo scheduling dalla coda verso la testa, esso prima di tutto inserirà J_4, poi tra J_2 e J_3 sceglierà J_2 perché possiede la deadline maggiore, infine tra J_1 e J_3 inserirà prima J_1 e poi J_3. Lo schedule risultante è quindi J_3, J_1, J_2, J_4.



4) Prendiamo il seguente insieme di task:

Task	T	C	P	U
1	8	3	1	$\frac{3}{8}$
2	8	4	2	$\frac{1}{2}$
3	8	1	3	$\frac{1}{8}$

per cui risulta $U = \frac{3}{8} + \frac{1}{2} + \frac{1}{8} = 1 \not\leq U(3) = 0,78$.
Ma tale insieme di task è comunque schedulabile con RM come si vede nel seguente diagramma temporale:

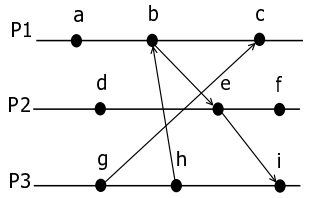


5) L'esercizio riguarda argomenti non pi trattati nel corso.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
25 Settembre 2002

---

1. Numerare gli eventi del seguente diagramma con i corrispondenti timestamp sia nel caso di clock logici, sia nel caso di *vettori* di clock logici.



[4 punti]

2. Spiegare il concetto di serializzabilità di un insieme di transazioni, e mostrare come il protocollo di *commit a due fasi* garantisce la serializzabilità di un insieme di transazioni. **[6 punti]**

3. Descrivere il protocollo di modifica basato su *voting* per l'accesso ad un file server da parte di un client. **[5 punti]**

4. Si consideri il seguente insieme di task periodici:

Task	T	C
1	20	14
2	40	C_2
3	80	4
4	100	5

Supponendo di voler costruire per questo insieme di task uno schedule *statico*, per quale valori di C_2 ciò è possibile? Perchè? **[8 punti]**

5. Si consideri il seguente insieme di task periodici schedulati con RM, che non soddisfa il test di utilizzazione della CPU.

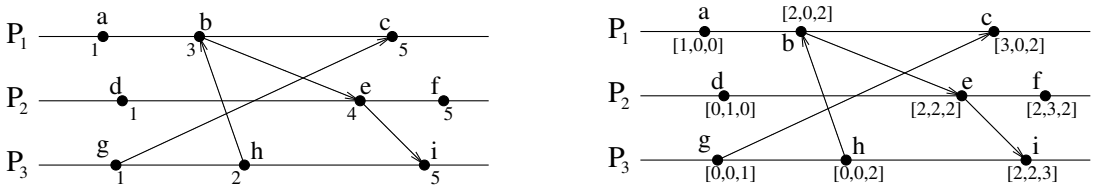
Task	*Priorità*	C_i	T_i
τ_1	+++	3	6
τ_2	++	3	13
τ_3	+	5	18

Si dimostri usando l'analisi dei tempi di risposta (R_i) che l'insieme è comunque schedulabile, e mostrare lo schedule corrispondente.

[5 punti]

SOLUZIONE:

1) Vediamo in Figura 3(a) il diagramma risultante dall'applicazione dei clock logici e in Figura 3(b) il diagramma risultante nel caso di utilizzo dei vettori di clock logici.



(a) (b)

Figura 3: soluzione all'esercizio numero 1

2) L'esercizio riguarda argomenti non pi trattati nel corso.

3) L'algoritmo di Gifford basato su voting per la modifica di un file replicato segue i seguenti passi:

- un client per poter accedere ad un file replicato deve prima ottenere il benestare da parte di un certo numero di server;
- ad ogni copia è associato un numero di versione che viene aggiornato ad ogni scrittura;
- il client chiede il permesso di accedere al file ai server che ne possiedono una copia (poniamo che siano N);
- il client otterrà risposta riceverà il permesso di lettura da un certo numero, diciamo r (read quorum), di server e il permesso di scrittura da un altro insieme di server, diciamo w (write quorum).
- allora $r + w > N$ e almeno una delle copie del quorum di lettura sarà aggiornata (quella con il numero di versione maggiore)

4) Osservando i periodi dei diversi task si può calcolare che il minor cycle dura 20 istanti di tempo mentre la durata del major cycle è di 400 (minimo comune multiplo dei diversi periodi).
Il testo dell'esercizio non specifica se lo schedule statico da utilizzare debba essere preemtive o no, risolviamo quindi l'esercizio in entrambi i modi.

a. Nel caso preemptive l'esecuzione dei diversi task può essere suddivisa in diversi minor cycles purché rispetti le deadlines.
Il massimo valore di C_2 si ottiene allora "impacchettando" in modo ottimale gli altri 3 task e vedendo quanto tempo di CPU rimane libero.
Il task 1 deve essere eseguito in ogni minor cycle quindi dei 20 istanti del minor cycle rimangono ora liberi solo 6 istanti. Il task 3 può essere suddiviso equamente in 4 minor cycle lasciando così liberi 5 istanti. Infine l'esecuzione del quarto task può essere suddivisa i 5 minor cycles. Il massimo valore di C_2 è quindi 4.

b. Analogamente al punto a. nel caso non preemptive si devono impacchettare in modo ottimale i tre task tenendo però conto che l'esecuzione non può essere suddivisa in diversi minor cycles.
Come prima il task 1 richiede 14 istanti per ogni minor cycle. Il terzo task richiede 4 istanti d'esecuzione ogni 80, poniamo quindi di eseguirlo nei minor cycle che partono agli istanti 80*t. Il 4 quarto task deve eseguire per 5 istanti ogni 100 e lo eseguiamo nel 2, 7, 11 e 19 minor cycle. Rimangono liberi quindi 6 istanti nei minor cycles di indice pari. Il massimo valore di C_2 è quindi 6.

5) Il testo dell'esercizio è errato dato che utilizzando i valori dati risulta un utilizzo totale maggiore di 1 e questo implica che l'insieme di task è sicuramente non schedulabile.
Consideriamo invece tale esercizio con C_3=3. Con tale valore si ha che U=0,897>U(3)=0,78.
Vediamo però utilizzando l'analisi dei tempi di risposta, che l'insieme è comunque schedulabile.

Task 1: $R_1 = C_1 = 3 \leq 6$ ✓

Task 2: $w_2^0 = C_2 = 3$

$w_2^1 = 3 + \lceil 3/6 \rceil * 3 = 6$

$w_2^2 = 3 + \lceil 6/6 \rceil * 3 = 6 \quad \Rightarrow \quad R_2 = 6 \leq 13$ ✓

Task 3: $w_3^0 = C_3 = 3$

$w_3^1 = 3 + \lceil 3/6 \rceil * 3 + \lceil 3/13 \rceil * 3 = 9$

$w_3^2 = 3 + \lceil 9/6 \rceil * 3 + \lceil 9/13 \rceil * 3 = 12$

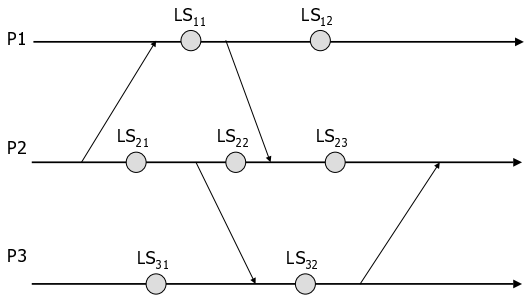
$w_3^3 = 3 + \lceil 12/6 \rceil * 3 + \lceil 12/13 \rceil * 3 = 12 \quad \Rightarrow \quad R_3 = 12 \leq 18$ ✓

L'insieme dei task è quindi schedulabile.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
3 Dicembre 2002

---

1. Si consideri l'insieme di tre processi P_1, P_2 e P_3 indicato in figura, le frecce indicano l'invio di un messaggio, ed i cerchi uno stato *locale* di ogni processo. Per ogni possibile stato globale costituito da una terna di stati locali $GS = (LS_{1i}, LS_{2j}, LS_{3k})$, $\forall i, j, k$, si determini se è uno stato consistente, e, in caso contrario si indichi il perchè.



NOTA: Ci sono $12 = 2 \cdot 3 \cdot 3$ possibili stati globali. **[5 punti]**

2. Si consideri il seguente insieme di task a, b, c, d, che devono accedere a due risorse condivise (semafori) P e Q. I task hanno le seguenti caratteristiche:

Task	*Priorità*	*Tempo di rilascio*	*Sequenza di operazioni*
a	4	4	EEQVE
b	3	2	EVVE
c	2	2	EE
d	1	0	EQQQQE

Le operazioni possibili sono 3: E (esecuzione), P (accesso a P) e V (accesso a V). Ogni istanza di operazione indica un'unità di tempo. Per esempio, EVVE = (E per un'unità di tempo, V per due unità di tempo, etc.).

Mostrare l'esecuzione dei 4 processi nei tre casi:

(a) Utilizzo di un normale algoritmo a priorità (4=max, 1=min);

(b) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando la *priority inheritance*;

(c) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando il *priority ceiling*;

Si mostrino i diagrammi temporali, e si calcoli per ogni task il tempo di risposta.

[8 punti]

3. Si consideri il seguente insieme di task periodici:

Task	T	C
1	25	14
2	25	C_2
3	50	4
4	100	5

Supponendo di voler costruire per questo insieme di task uno schedule *statico*, per quale valori di C_2 ciò è possibile? Si considerino i due casi distinti:

a) I task debbano essere eseguiti tutti in modo non preemptive;

b) Il task 2 può essere eseguito suddividendolo in **al massimo** due sottotask.

[5 punti]

4. Costruire un insieme di quattro task periodici che sia schedulabile, e per il quale il test di utilizzazione basato su U fallisca.

NOTA: $U(4) \approx 0.757$. **[4 punti]**

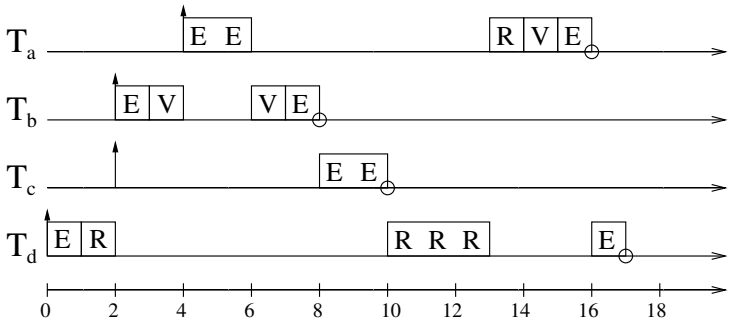
5. Descrivere il funzionamento di principio di NFS, ed in particolare i principali processi coinvolti e l'architettura software. **[4 punti]**

SOLUZIONE:

1) Uno stato si dice consistente se per ogni evento che include, esso include anche tutti gli eventi che lo precedono causalmente. I 12 stati globali possibili possono allora essere così classificati:

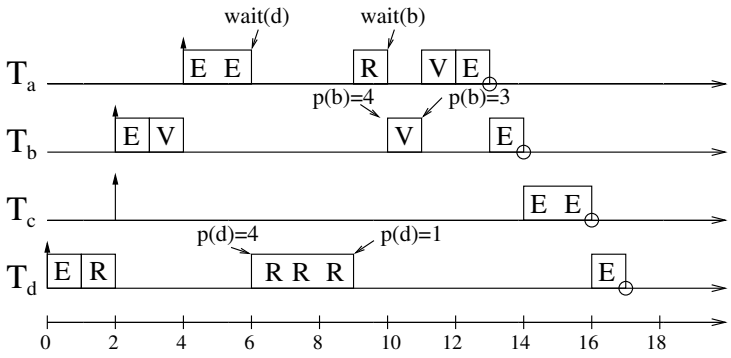
- $(LS_{11},\ LS_{21},\ LS_{31})$ è consistente
- $(LS_{11},\ LS_{21},\ LS_{32})$ non è consistente perché include (R, m_2) ma non (S, m_2)
- $(LS_{11},\ LS_{22},\ LS_{31})$ è consistente
- $(LS_{11},\ LS_{22},\ LS_{32})$ è consistente
- $(LS_{11},\ LS_{23},\ LS_{31})$ non è consistente perché include (R, m_3) ma non (S, m_3)
- $(LS_{11},\ LS_{23},\ LS_{32})$ non è consistente perché include (R, m_3) ma non (S, m_3)
- $(LS_{12},\ LS_{21},\ LS_{31})$ è consistente
- $(LS_{12},\ LS_{21},\ LS_{32})$ non è consistente perché include (R, m_2) ma non (S, m_2)
- $(LS_{12},\ LS_{22},\ LS_{31})$ è consistente
- $(LS_{12},\ LS_{22},\ LS_{32})$ è consistente
- $(LS_{12},\ LS_{23},\ LS_{31})$ è consistente
- $(LS_{12},\ LS_{23},\ LS_{32})$ è consistente

2) a.



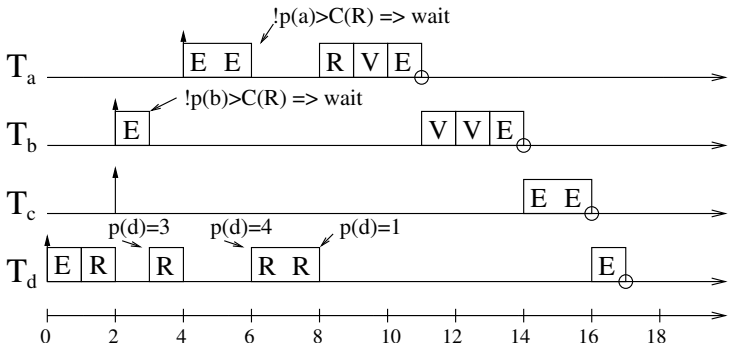
I tempi di risposta risultano quindi essere:
$T_{r_a} = 12 \qquad T_{r_b} = 6 \qquad T_{r_c} = 8 \qquad T_{r_d} = 17 \qquad \overline{T}_r = 10,75$

b.



I tempi di risposta risultano quindi essere:
$T_{r_a} = 9 \qquad T_{r_b} = 12 \qquad T_{r_c} = 14 \qquad T_{r_d} = 17 \qquad \overline{T}_r = 13$

c. Al fine di poter applicare il priority ceiling è necessario definire le priorità delle risorse R e V: C(R)=4 e C(V)=4.



I tempi di risposta risultano quindi essere:
$T_{r_a} = 7 \qquad T_{r_b} = 12 \qquad T_{r_c} = 14 \qquad T_{r_d} = 17 \qquad \overline{T}_r = 12,5$

3) a. Un modo per trovare il valore di C_2 è quello di schedulare al meglio gli altri task e poi valutare il tempo rimasto libero.
Osservando i periodi dei task è possibile calcolare che il minor cycle è lungo 25 mentre il major cycle è lungo 100 ed è quindi composto da 4 minor cycle.
T_1 è presente in ogni minor cycle e questo lascia liberi 11 istanti in ogni minor cycle. T_3 deve essere eseguito ogni 2 cicli, poniamo per esempio il primo ed il terzo, lasciando così liberi 7, 11, 7 e 11 istanti nei quattro cicli. Infine T_4 verrà eseguito in uno solo dei 4 cicli che compongono un major cycle, poniano il secondo. Nei 4 minor cycles rimangono quindi rispettivamente 7, 6, 7 e 11 istanti liberi. Dato che T_2 deve essere eseguito in ogni minor cycle dobbiamo prendere il valore minore tra i quattro sopra indicati e quindi risulta C_2=6.

b. Anche in questo caso applichiamo lo stesso ragionamento del punto (a). Iniziamo però con il notare che T_2 deve obbligatoriamente essere eseguito in ogni minor cycle e quindi non è affatto richiesto di suddividerlo.
Dato che ora i task 3 e 4 possono essere suddivisi come si vuole il modo migliore per schedularli è quello di spezzarli equamente tra i minor cycles.
Ogni minor cycle prevederà quindi 14 istanti per T_1, 2 istanti (4 istanti ogni 2 cicli) per T_3 e 1,25 istanti (5 istanti ogni 4 cicli) per T_4.
In ogni minor cycle rimangono allora liberi 25-14-2-1,25=7,75 istanti e si avrà C_2=7,75.

4) Prendiamo il seguente insieme di task:

Task	T	C	Priorità	U
1	80	40	1	0.5
2	40	10	2	0.25
3	20	2.5	3	0.125
4	20	2.5	4	0.125

per cui risulta $U = 1 \not\leq U(4) = 0,757$.
L'insieme dei tempi di task è comunque schedulabile come mostra la Response Time Analisys:

Task 4: $R_4 = C_4 = 2.5 \leq 20$ ✓

Task 3: $w_3^0 = C_3 = 2.5$
$w_3^1 = 2.5 + \lceil 2.5/20 \rceil * 2.5 = 5$
$w_3^2 = 2.5 + \lceil 5/20 \rceil * 2.5 = 5 \quad \Rightarrow \quad R_3 = 5 \leq 20$ ✓

Task 2: $w_2^0 = C_2 = 10$
$w_2^1 = 10 + \lceil 10/20 \rceil * 2.5 + \lceil 10/20 \rceil * 2.5 = 15$
$w_2^2 = 10 + \lceil 15/20 \rceil * 2.5 + \lceil 15/20 \rceil * 2.5 = 15 \quad \Rightarrow \quad R_2 = 15 \leq 40$ ✓

Task 1: $w_1^0 = C_1 = 40$
$w_1^1 = 40 + \lceil 40/40 \rceil * 10 + \lceil 40/20 \rceil * 2.5 + \lceil 40/20 \rceil * 2.5 = 60$
$w_1^2 = 40 + \lceil 60/40 \rceil * 10 + \lceil 60/20 \rceil * 2.5 + \lceil 60/20 \rceil * 2.5 = 65$
$w_1^3 = 40 + \lceil 65/40 \rceil * 10 + \lceil 65/20 \rceil * 2.5 + \lceil 65/20 \rceil * 2.5 = 80$
$w_1^4 = 40 + \lceil 80/40 \rceil * 10 + \lceil 80/20 \rceil * 2.5 + \lceil 80/20 \rceil * 2.5 = 80 \quad \Rightarrow \quad R_1 = 80 \leq 80$ ✓

5) NFS è basato su un'architettura client/server, anche se la distinzione tra i due è puramente concettuale dato che essi eseguono lo stesso software.
Si chiama server un host che mette a disposizione (esporta)una o pi parti del prorpio file system. I file system epsortati devono essere registrati nel file /etc/exports. Il server richiede in esecuzione 3 demoni:

- nfsd che accetta le richieste dai client e le passa a mountd;
- mountd che gestisce le richieste;
- rpcbind che consente ai client di conoscere la porta su cui è in attesa il server NFS.

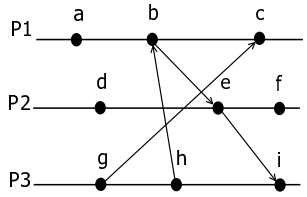
I client possono integrare nel proprio file system locale i file system del server eseguendone il mount in un mount point locale. È inoltre possibile eseguire l'automounting all'avvio del client registrando opportunamente i file system nel file /etc/fstab.
Il protocollo di mount prevede che il client invii una richiesta al server contenente il path del file system a cui è interessato. Il server verifica che il path sia esportato e in caso affermativo ritorna al client un file handle che verrà utilizzato dal client per tutte le operazioni successive.
NFS prevede anche un protocollo per l'accesso a file contenente delle primitive per tutte le operazioni che possono essere eseguite sui file.
NFS è basato su RPC e di conseguenza è stateless ed indipendente dal protocollo di trasporto utilizzato dalla rete sottostante.
L'architettura NFS è divisibile in tre strati:

1. system call layer;
2. Virtual File System (VFS) layer: mantiene una tabella di v-node, uno per ogni file aperto. Ogni v-node punta ad un i-node se il file è locale o ad un r-node se è remoto;
3. NFS layer: traduce le richieste secondo il protocollo NFS.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
17 Dicembre 2002

---

1. Numerare gli eventi del seguente diagramma con i corrispondenti timestamp sia nel caso di clock logici, sia nel caso di *vettori* di clock logici.



[4 punti]

2. Si consideri il seguente insieme di task **aperiodici**:

Task	a_i	C_i	d_i
J_1	4	4	10
J_2	3	5	12
J_3	2	2	14
J_4	0	3	16

Si scheduli inizialmente l'insieme di task usando EDF.

Successivamente, si supponga di diminuire tutte le deadline di una quantità Δ. Calcolare il massimo valore di Δ per cui l'insieme resta schedulabile.

Partendo infine dall'analisi precedente, determinare un possibile criterio (per es. una formula) per determinare la schedulabilità di un insieme di task **aperiodici** a priori, simile al test basato su U (nel caso di task **periodici**.

SUGGERIMENTI:

- il criterio deve in qualche modo tenere conto dei tempi di arrivo, per esempio considerando le deadline relative $D_i = d_i - a_i$;
- per il motivo precedente, la formula deve essere funzione del tempo.

NOTA: *E' importante il ragionamento, più che la correttezza formale della formula!*

[9 punti]

3. Si consideri il seguente insieme di task periodici, che non soddisfa il test di utilizzazione della CPU.

Task	*Priorità*	C_i	T_i
τ_1	+++	3	6
τ_2	++	3	13
τ_3	+	5	18

Si dimostri usando l'analisi dei tempi di risposta (R_i) che l'insieme è comunque schedulabile, e mostrare lo schedule corrispondente.

[5 punti]

4. Descivere il funzionamento dell'algoritmo di Ricart & Agrawala per la gestione distribuita della sincronizzazione tra un insieme di processi, mostrando anche un esempio di funzionamento.

   Si evidenzino inoltre vantaggi e svantaggi rispetto ad un approccio centralizzato. **[4 punti]**

5. Descrivere le caratteristiche principali e mostrare le principali differenze tra S.O. basati su microkernel ed exokernel

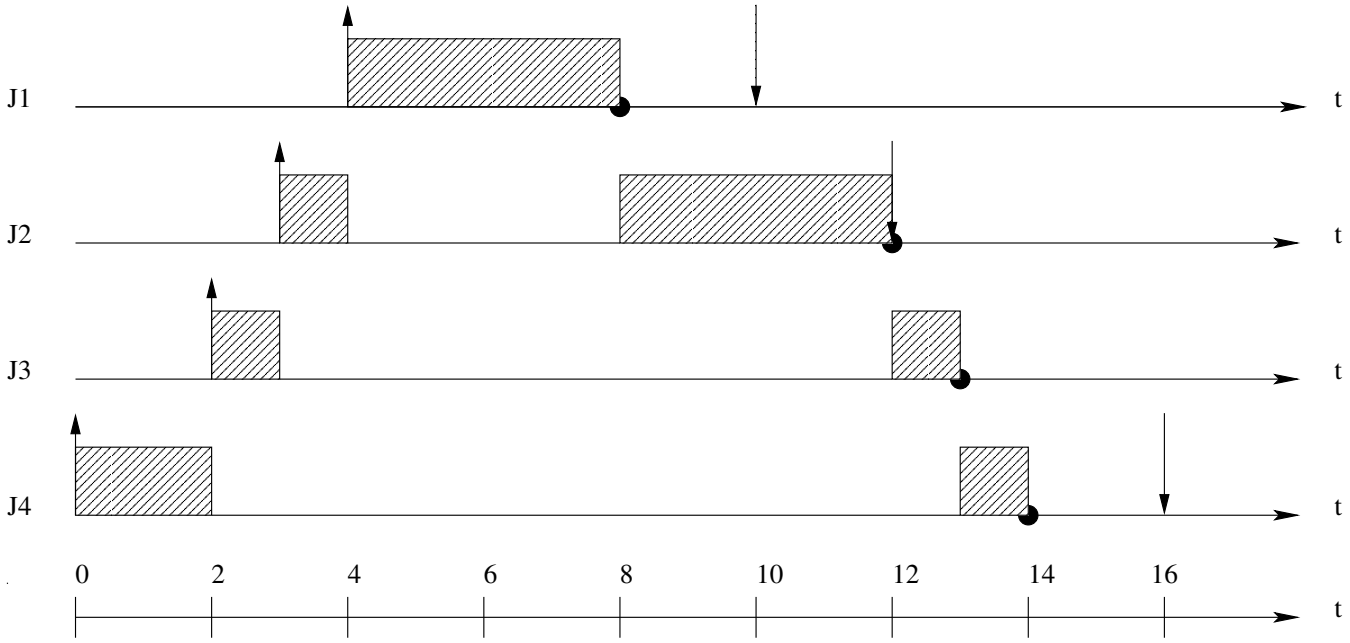
[4 punti]

SOLUZIONE:

1) Vedi esercizio 1 del 25 Settembre 2002

2)

a. In EDF i task vengono ordinati secondo deadline crescente, ma se nell'esecuzione di un task ne arriva uno con deadline pi vicina prelazione quello in esecuzione. Lo scheduling che si ottiene è il seguente:



EDF schedula correttamente l'insieme dei task.

b. Il massimo valore di Δ è zero perché se tutti i task diminuiscono le loro deadline nella stessa misura avremo sempre le stesse prelazioni (in quanto EDF lavorerebbe sempre allo stesso modo) e J_2 non riuscirebbe pi a completare: $\Delta = min(D_i - f_i) = min(2; 0; 1; 2) = 0$

3) Vedi esercizio 5 del 25 Settembre 2002

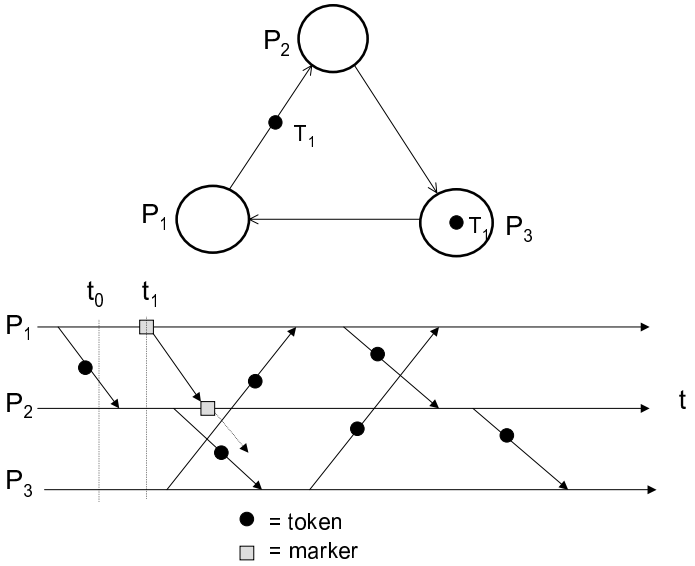
4) Vedi esercizio 1 del 10 Luglio 2002

5) Vedi esercizio 1 del 19 Giugno 2002

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
26 Marzo 2003

---

1. Si considerino i tre processi P_1, P_2, P_3 indicati in figura.



I tre processi comunicano tramite canali unidirezionali, e la comunicazione consiste nello scambio di due *token* T_1 e T_2, inizialmente (al tempo T_0) nella situazione indicata nella figura (cioè T_1 è in possesso di P_3 e T_2 è in transito tra P_1 e P_2.

Si supponga poi che l'esecuzione del sistema evolva come indicato dal diagramma temporale, in cui i pallini neri indicano trasmissione di token.

Si mostri l'esecuzione dell'algoritmo dei *distributed snapshot*, indicando il risultato da esso fornito, assumendo che l'algoritmo sia lanciato da P_1 al tempo t_1 (vedi figura).

Giustificare la risposta **[9 punti]**

2. Si consideri il seguente insieme di task a, b, c, d, che devono accedere a due risorse condivise (semafori) P e Q. I task hanno le seguenti caratteristiche:

Task	*Priorità*	*Tempo di rilascio*	*Sequenza di operazioni*
a	4	4	EEQVE
b	3	2	EVVQQE
c	2	2	EEE
d	1	0	EQQQE

Le operazioni possibili sono 3: E (esecuzione), P (accesso a P) e Q (accesso a Q). Ogni istanza di operazione indica un'unità di tempo. Per esempio, EVVE = (E per un'unità di tempo, V per due unità di tempo, etc.).

Mostrare l'esecuzione dei 4 processi nei tre casi:

(a) Utilizzo di un normale algoritmo a priorità (4=max, 1=min);

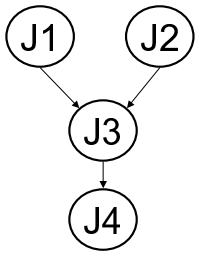
(b) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando la *priority inheritance*;

(c) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando il *priority ceiling*;

Si mostrino i diagrammi temporali, e si calcoli per ogni task il tempo di risposta.

[8 punti]

3. Sia dato il seguente insieme di task aperiodici, legati dal seguente grafo delle dipendenze:



Task	C_i	d_i
T_1	3	5
T_2	2	6
T_3	1	3
T_4	3	9

Fornire una schedulazione valida dei task utilizzando l'algoritmo EDF*, modificando opportunamente i tempi di rilascio e le deadline. [**6 punti**]

4. Si consideri il seguente insieme di task periodici:

Task	C_i	T_i
τ_1	2	6
τ_2	2	10
τ_3	1	12

Si consideri poi la seguente sequenza di richieste aperiodiche:

$$(1,2),(6,1),(10,1),(15,2),(22,1)$$

dove (i,j) indicano rispettivamente il tempo di arrivo a_i e il burst c_i. Si assuma che la deadline di ogni richiesta aperiodica corrisponda al tempo di arrivo della richiesta successiva. Per esempio, la richiesta $(2,2)$ ha come deadline il tempo $t = 6$.

Si mostri l'andamento temporale dei vari processi e delle richieste aperiodiche usando una soluzione di tipo *background scheduling*, in cui la coda dei processi periodici è gestita secondo RM, e quella delle richieste aperiodiche in modo FIFO.

Si indichi chiaramente se tutte le deadline vengono rispettate. [**7 punti**]

SOLUZIONE:

1) L'algoritmo dei distributed snapshot è basato sull'utilizzo di un particolare messaggio (marker) che funge da elemento di sincronizzazione.
Perché l'algoritmo funzioni è necessario però che i canali di comunicazione siano di tipo FIFO e che non ci possano essere perdita di messaggi.
L'algoritmo è avviato da un processo che, dopo aver registrato il proprio stato locale, invia il marker su tutti i canali d'uscita. Un generico processo invece si comporta come segue:

- alla prima ricezione del marker su un canale il processo registra il proprio stato indicando come vuoto il canale su cui ha ricevuto il marker. Infine propaga il marker su tutti i suoi canali d'uscita e si mette in attesa sui suoi canali d'ingresso.
- alle successive ricezioni del marker su un certo canale il processo registra lo stato del canale indicando la sequenza di messaggi ricevuti attraverso di esso a partire dalla prima ricezione del marker.

L'algoritmo termina quando tutti i processi hanno ricevuto il marker su tutti i canali d'ingresso.
È importante notare come la raccolta delle informazioni registrate dai diversi processi richieda un ulteriore processo.
Applicando l'algoritmo alla rete dell'esercizio si ottiene il seguente stato globale:

1

Node	Recorded State			
	State	c_1	c_2	c_3
P_1	$\emptyset$			
P_2				
P_3				

2

Node	Recorded State			
	State	c_1	c_2	c_3
P_1	$\emptyset$			
P_2	$\emptyset$	{ }		
P_3				

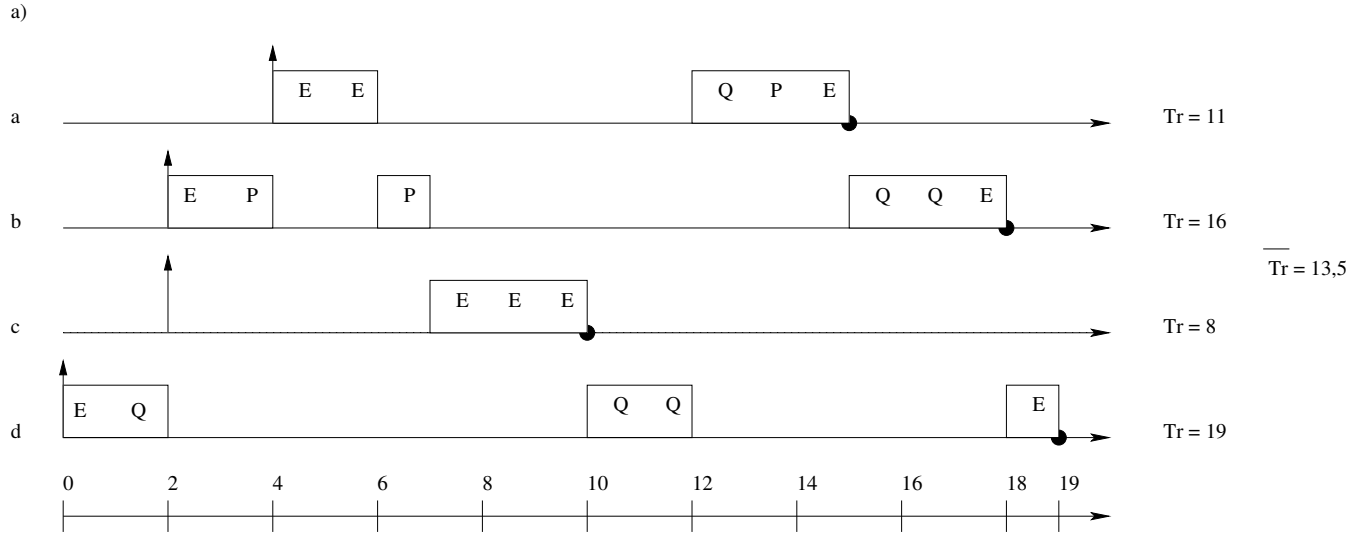
3

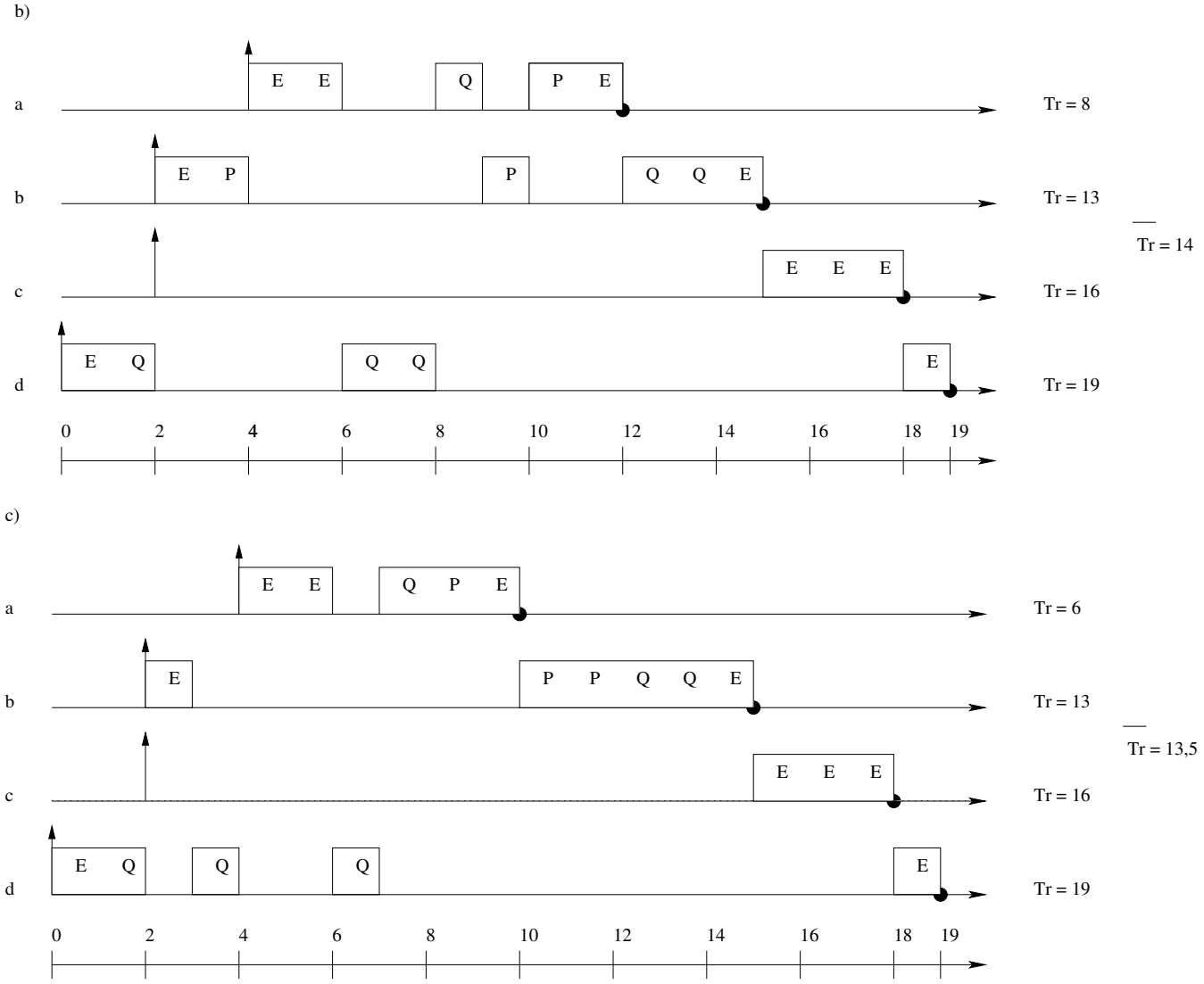
Node	Recorded State			
	State	c_1	c_2	c_3
P_1	$\emptyset$			
P_2	$\emptyset$	{ }		
P_3	•		{ }	

4

Node	Recorded State			
	State	c_1	c_2	c_3
P_1	$\emptyset$			•
P_2	$\emptyset$	{ }		
P_3	•		{ }	

2) Il tempo di risposta viene calcolato con la seguente formula: $T_r = f_i - r_i$

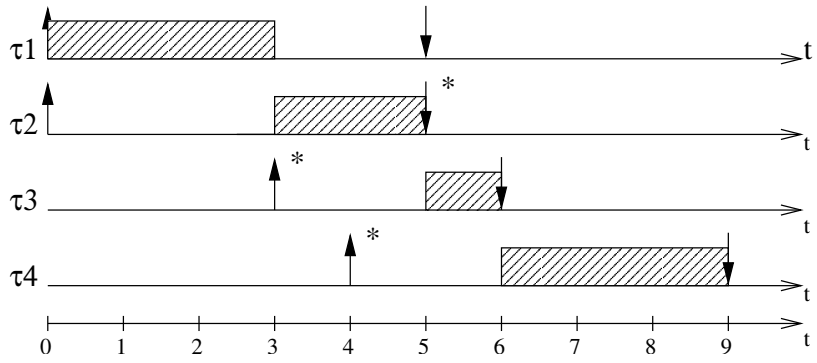




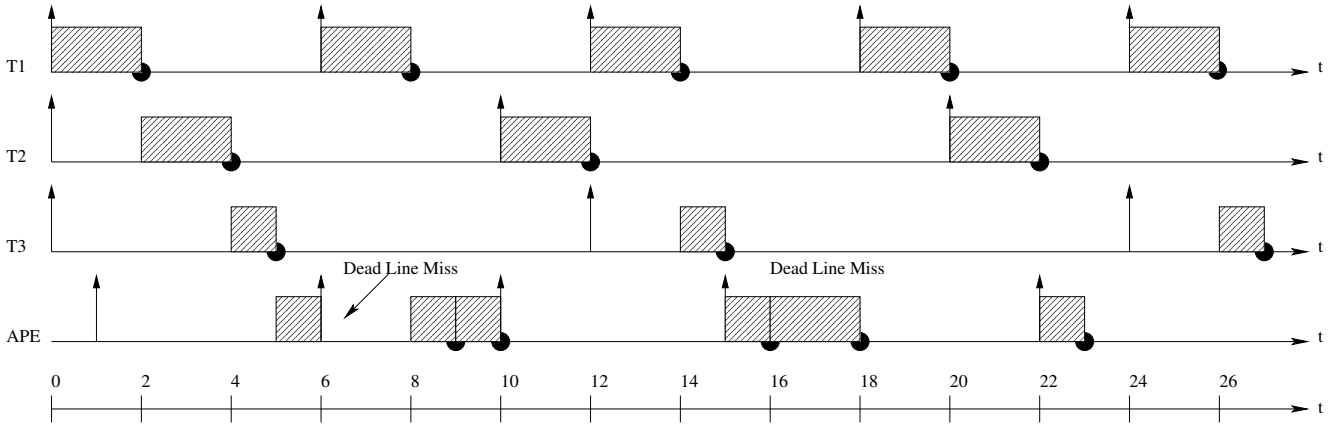
3) Il testo di questo esercizio viene leggermente modificato per consentire un corretto svolgimento: $d_3 = 6$ anziché 3. Ora applicando le formule viste a lezione i tempi di rilascio e deadline cambiano così

- $d_2 = 5$
- $a_3 = 3$
- $a_4 = 4$

Applicando EDF* si ottiene il seguente diagramma temporale:



4) Applicando l'algoritmo background scheduling si ottiene il seguente diagramma temporale:

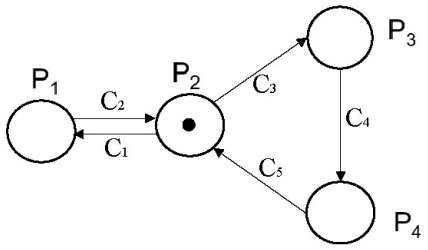


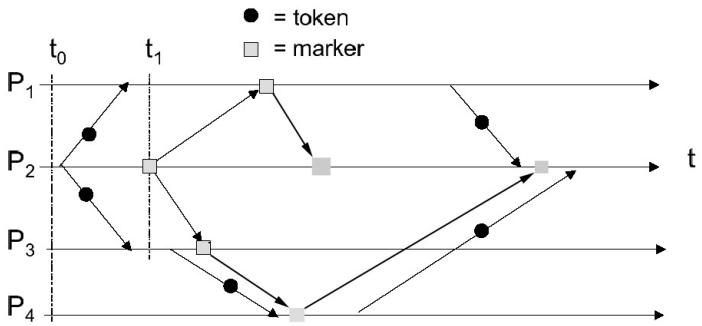
Si può notare che la sequenza di task aperiodica non viene servita rispettando le deadline. È ipotizzabile un risultato di questo genere in quanto l'algoritmo utilizzato prevede l'utilizzo di due code, una per i processi periodici (ad alta priorità ed una per quelli aperiodici (a bassa priorità e perciò il tempo di risposta per i processi aperiodici può essere anche molto alto.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
27 Maggio 2003

---

1. Si considerino i quattro processi P_1, P_2, P_3, P_4 indicati in figura.





I quattro processi comunicano tramite canali unidirezionali, e la comunicazione consiste nello scambio di un *token* T, inizialmente (al tempo t_0) nella situazione indicata nella figura (cioè T è in possesso di P_2).

Si supponga poi che l'esecuzione del sistema evolva come indicato dal diagramma temporale, in cui i pallini neri indicano trasmissione di token.

Si mostri l'esecuzione dell'algoritmo dei *distributed snapshot*, indicando il risultato da esso fornito, assumendo che l'algoritmo sia lanciato da P_2 al tempo t_1 (vedi figura).

Giustificare la risposta **[9 punti]**

2. Si considerino due task τ_1 e τ_2, aventi il seguente profilo temporale:

Task	*Tempo di rilascio*	*Deadline*	*Sequenza di operazioni*
τ_1	2	8	EESSEE
τ_2	0	22	EESSSSSSSSSSSSEE

Le operazioni possibili sono 2: E (esecuzione) e S, che rappresenta l'accesso ad una risorsa in mutua esclusione . Ogni istanza di operazione indica un'unità di tempo.

Si supponga che i due task vengano inizialmente schedulati secondo EDF, e si mostri il relativo diagramma temporale.

Successivamente, si supponga che l'insieme di task venga eseguito su un processore a velocità doppia, e che quindi ciascuna delle operazioni occupi la metà del tempo. Mostrare il nuovo diagramma temporale, e si descriva il problema che si verifica.

Si spieghi quindi come sia possibile risolvere il problema corrispondente, e si mostri un diagramma temporale "corretto".

[8 punti]

3. Mostrare la sequenza di messaggi risultante dall'applicazione dell'algoritmo di elezione *bully*, nel caso di 6 processi, in seguito alla seguente sequenza di eventi:

   (a) L'attuale coordinatore P_6 va in crash;

   (b) Il processo P_3 lancia per primo un'elezione.

   Si descriva brevemente il principio usato dall'algoritmo. **[4 punti]**

4. Descrivere le possibili soluzioni per l'implementazione della memoria condivisa distribuita (DSM), indicandone le caratteristiche salienti.

   [4 punti]

5. Si consideri il seguente insieme di task periodici:

Task	T	C
1	20	3
2	10	C_2
3	12	1
4	8	2

   Si supponga che l'insieme di task sia schedulato usando RM. Indicare qualè il valore massimo di C_2 per cui l'insieme è schedulabile. Si confronti la soluzione ottenuta con il fattore di utilizzazione U ($U(4) = 0.757$) con quella ottenuta graficamente.

   Successivamente, si assuma $C_2 = 1$, e si supponga poi che tutti i valori C_i vengano aumentati di un fattore α. Siano $C_i' = (1 + \alpha)C_i$ i nuovi valori di C_i. Si valuti analiticamente la schedulabilità dell'insieme come funzione di α.

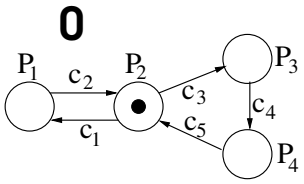
   [6 punti]

SOLUZIONE:

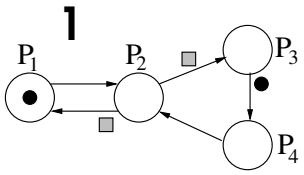
1) L'algoritmo dei distributed snapshot è basato sull'utilizzo di un particolare messaggio (marker) che funge da elemento di sincronizzazione.
Perché l'algoritmo funzioni è necessario però che i canali di comunicazione siano di tipo FIFO e che non ci possano essere perdita di messaggi.
L'algoritmo è avviato da un processo che, dopo aver registrato il proprio stato locale, invia il marker su tutti i canali d'uscita. Un generico processo invece si comporta come segue:

- alla prima ricezione del marker su un canale il processo registra il proprio stato indicando come vuoto il canale su cui ha ricevuto il marker. Infine propaga il marker su tutti i suoi canali d'uscita e si mette in attesa sui suoi canali d'ingresso.
- alle successive ricezioni del marker su un certo canale il processo registra lo stato del canale indicando la sequenza di messaggi ricevuti attraverso di esso a partire dalla prima ricezione del marker.

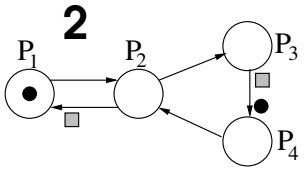
L'algoritmo termina quando tutti i processi hanno ricevuto il marker su tutti i canali d'ingresso.
È importante notare come la raccolta delle informazioni registrate dai diversi processi richieda un ulteriore processo.
Applicando l'algoritmo alla rete dell'esercizio si ottiene quanto seguente:



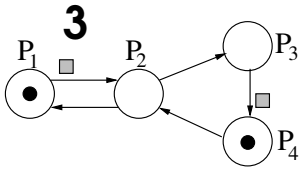
Nodo	Stato registrato					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1		{ }				
P_2			{ }			{ }
P_3				{ }		
P_4					{ }	



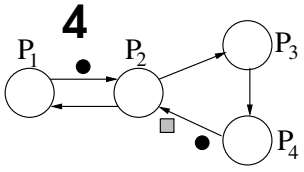
Nodo	Stato registrato					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1		{ }				
P_2	∅		{ }			{ }
P_3				{ }		
P_4					{ }	



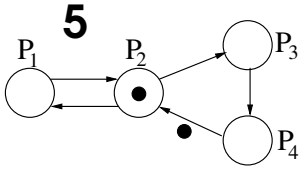
Nodo	Stato registrato					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1		{ }				
P_2	∅		{ }			{ }
P_3	∅			{ }		
P_4					{ }	



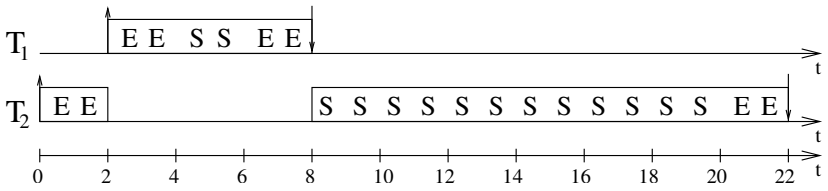
Nodo	Stato registrato					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1	•	{ }				
P_2	∅		{ }			{ }
P_3	∅			{ }		
P_4					{ }	



Nodo	Stato registrato					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1	•	{ }				
P_2	∅		{ }			{ }
P_3	∅			{ }		
P_4	•				{ }	

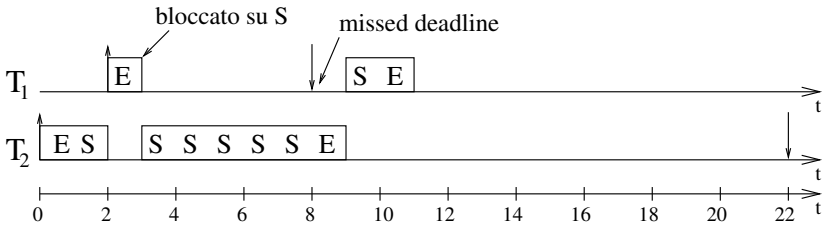


Nodo	Stato registrato finale					
	Stato	c_1	c_2	c_3	c_4	c_5
P_1	•	{ }				
P_2	∅		{ }			{ }
P_3	∅			{ }		
P_4	•				{ }	



2) a. Il diagramma temporale che si ottiene utilizzando EDF è il seguente:

b. Ora T_2 prima che arrivi T_1 ha il tempo di entrare nella sezione critica, quindi T_1 quando al tempo 3 cerca a

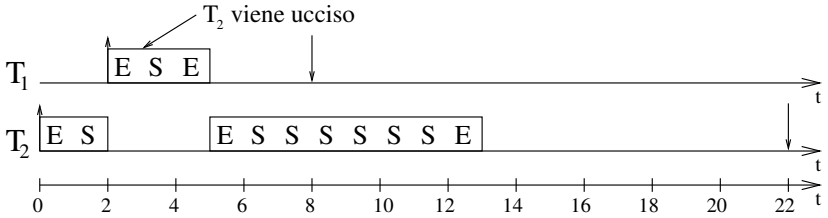


sua volta di entrarvi rimane bloccato. A questo punto T_2 torna in esecuzione fino a quando non esce dalla sezione critica, ma a quel punto T_1 ha già superato la deadline.

c. La soluzione è utilizzare un algoritmo di gestione dei deadlock. In particolare si usa Wound-Wait basato sulle deadline invece che sul timestamp. Quando un processo P_1 necessita di una risorsa detenuta da un processo P_j:

- se $D_i < D_j$ $\Rightarrow$ P_j viene ucciso;
- se $D_i > D_j$ $\Rightarrow$ P_i si mette in attesa.

Allora il diagramma temporale risultante è il seguente:



3) Applicando l'algoritmo di elezione bully si ottiene la seguente sequenza di messaggi:

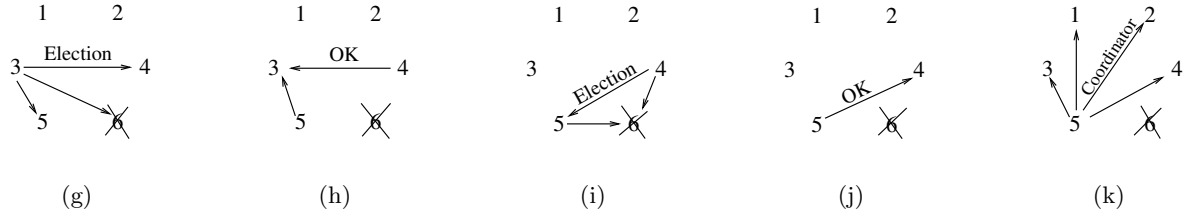


Figura 4: soluzione all'esercizio numero 3

4) Gli approcci per l'implementazione della memoria condivida distribuita possono essere classificati "incrociando" 2 dimensioni di progetto: la migrazione dei dati e la replicazione dei dati. Otteniamo quindi 4 possibili soluzioni.

	Senza replicazione	*Con replicazione*
Senza migrazione	Central server	Full replication
Con migrazione	Data migration	Read replication

Vediamole pi in dettaglio.

Central server: le pagine contenenti i dati sono mantenute tutte in un server centrale che riceve ed esegue le richieste dei client. Il server quindi ritorna i dati in caso di lettura ed effettua le modifiche in caso di scrittura. Al fine di evitare scritture multiple è necessario inoltre numerare le richieste.

Tale soluzione è semplice sia nell'implementazione che nella modalità di sincronizzazione utilizzata ma è poco efficiente dato che tutte le operazioni dei client sono remote, inoltre il server è un punto critico di rottura e può essere soggetto ad un sovraccarico di lavoro.

Data migration: è una soluzione basata sull'idea che i dati dovrebbero risiedere dove sono utilizzati pi spesso ed implementa un protocollo di tipo "lettore singolo/ scrittore singolo". Quando un host fa riferimento a dati remoti questi vengono migrati presso il richiedente. Naturalmente a tal fine è necessario sapere dove le pagine risiedono in ogni istante, questo può essere fatto utilizzando un server apposito che mantenga tale informazione, tramite broadcast o mantenendo le informazioni direttamente nei vari host.

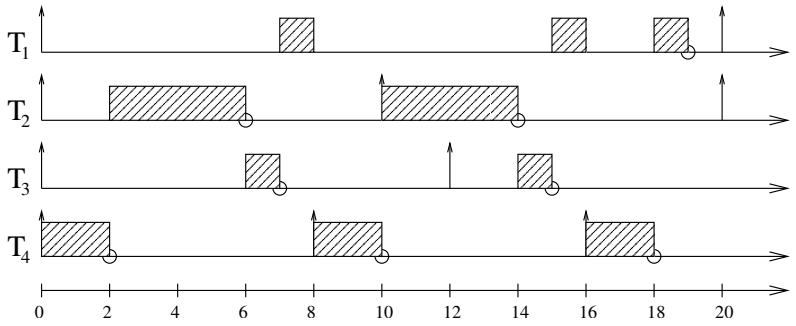
Questa soluzione riduce le richieste remote ma può introdurre un eccessivo overhead dovuto alla continua migrazione delle pagine. Questo problema può essere ridotto obbligando ogni pagina a risiedere in un host almeno un certo tempo dopo che è stata ricevuta.

Read replication: questa soluzione implementa un protocollo "lettore multiplo/ scrittore singolo". Ad ogni pagina è assegnato un possessore che la memorizza in modo permanente.
In lettura le pagine vengono replicate presso il richiedente (lettori multipli) ma in caso di scrittura tutte le copie devono essere invalidate e l'unica copia su cui avvengono effettivamente le modifiche è quel del possessore della pagina.
Anche in questo caso al fine di invalidare le diverse copie di una pagina è necessario tenerne traccia.

Tale soluzione può essere inefficiente dato che le scritture sono tutte remote, il possessore di una pagina popolare può essere sovraccaricato di lavoro ed inoltre è necessaria l'invalidazione di tutte le copie di una pagina in caso di una sua modifica.

Full replication: implementa un protocollo del tipo "lettore multiplo/ scrittore multiplo". Le diverse copie esistenti presso i diversi host possono essere accedute sia in lettura che in scrittura, ma dato che ci possono essere pi scritture contemporaneamente è necessario prevedere un meccanismo per mantenere la consistenza delle diverse copie. La soluzione pi semplice a questo fine è quella di utilizzare un sequencer che riceva e sequenzializzi, numerandole e propagandole, tutte le scritture di tutti i client.

5) a. Graficamente si ottiene che il massimo valore di C_2 per cui l'insieme dei task è schedulabile è 4, come si vede nel seguente diagramma temporale.
Calcolando invece C_2 come il valore per cui si ottiene un utilizzo uguale a U(4) si ottiene 2.7, questo mostra come la condizione basata sul fattore di utilizzo sia sufficiente ma non necessaria.



b. L'esercizio richiede di trovare il massimo valore di α per cui l'insieme di task è schedulabile. Tale valore può essere trovato andando per tentativi ed utilizzando il test di schedulabilità basato sull'analisi dei tempi di risposta. In particolare il task 4 non dà problemi dato che avendo la priorità pi elevata non subisce l'influenza degli altri tasks. Quindi per esso si trova che:

$$R_4=C_4^{'}=(1+\alpha)*2\leq 8 \qquad \Rightarrow \ \alpha \leq 3$$

Per il task 2 come mostra la tabella sottostante il massimo valore di α che lo rende schedulabile è 1,6.

α	iterazione 0	iterazione 1	iterazione 2	iterazione 3
1,5	2,5	7,5	7,5	7,5
1,6	2,6	7,8	7,8	7,8
1,7	2,7	8,1	13,5	13,5
1,8	2,8	8,4	14	14

Per il task 3 analogamente si trova che $\alpha \leq 1$.

α	iterazione 0	iterazione 1	iterazione 2	iterazione 3	iterazione 4
0,8	1,8	7,2	7,2	7,2	7,2
0,9	1,9	7,6	7,6	7,6	7,6
1	2	8	8	8	8
1,1	2,1	8,4	12,6	14,7	14,7

Infine per il task 1 deve essere $\alpha \leq 0{,}5$.

α	iterazione 0	iterazione 1	iterazione 2	iterazione 3	iterazione 4	iterazione 5	iterazione 6
0,4	4,2	9,8	12,6	15,4	15,4	15,4	15,4
0,5	4,5	10,5	15	16,5	19,5	19,5	19,5
0,6	4,8	11,2	16	17,6	20,8	22,4	22,4

Dunque il massimo valore di α per cui l'insieme di task risulta schedulabile è 0,5.

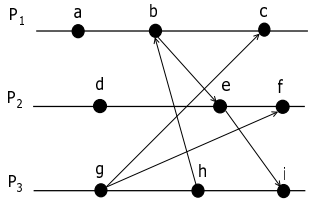
Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
25 Giugno 2003

---

1. Si illustrino i meccanismi di gestione dei deadlock in ambiente distribuito che utilizzano la *prevenzione statica* basata su *timestamp*. Si discutano le due possibili varianti, e si mostri un esempio. [**4 punti**]

---

2. Si consideri l'insieme di tre processi P_1, P_2 e P_3 indicato in figura, nel quale le frecce indicano l'invio di un messaggio, ed i cerchi uno stato *locale* di ogni processo



   Numerare gli eventi del seguente diagramma con i corrispondenti timestamp sia nel caso di clock logici, sia nel caso di *vettori* di clock logici. [**5 punti**]

---

3. Descrivere il protocollo usato da NFS per il *mount* di una directory remota [**4 punti**]

---

4. Descrivere i possibili modelli di consistenza implementabili in un sistema che utilizza la memoria condivisa distribuita (DSM), indicandone ove possibile un esempio. [**5 punti**]

---

5. Si considerino i seguenti due task periodici:

Task	T	C
1	20	5
2	10	3

   Si supponga che le deadline non coincidano con il periodo $D_i \neq T_i$, e che siano esprimibili come $D_i = \alpha \cdot T_i$, dove $0.5 \leq \alpha < 1$.

   Si calcoli il massimo valore di α per il quale l'insieme di task risulta schedulabile con DM ma non con RM.

   NOTE:

   - $U(2) = 0.828$
   - Non è obbligatorio applicare uno specifico test di schedulabilità (anche se semplifica i calcoli); è possibile procedere per tentativi.

   [**9 punti**]

SOLUZIONE:

1) Nella prevenzione statica dei deadlock basata sui timestamp ad ogni processo, al momento della sua creazione, viene associato un timestamp che rappresenta la priorità del processo stesso. Esistono 2 schemi basati sui timestamp:

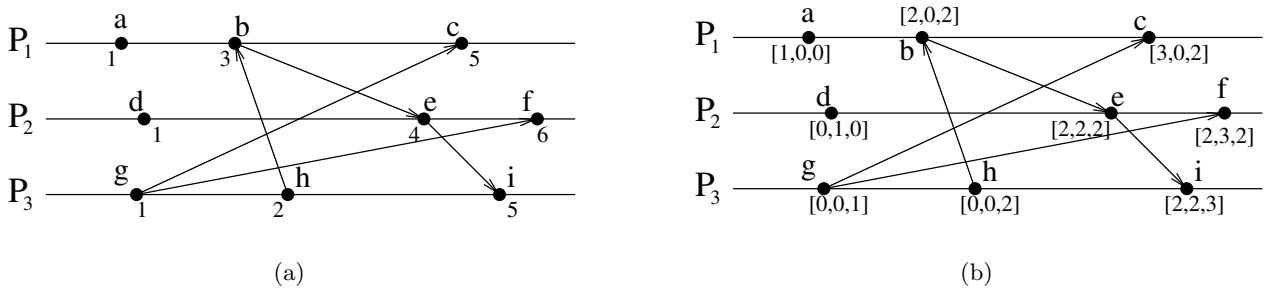
Wait-Die: è uno schema non preemptive. Quando un processo P_i necessita di una risorsa detenuta da un altro processo P_j:

- se $TS(P_i) < TS(P_j)$ allora P_i si mette in attesa (*wait*)
- se $TS(P_i) > TS(P_j)$ allora si esegue il rollback di P_i (*die*)

Wound-Wait: è uno schema preemptive. Quando un processo P_i necessita di una risorsa detenuta da un altro processo P_j:

- se $TS(P_i) < TS(P_j)$ allora P_j viene ucciso (*wound*)
- se $TS(P_i) > TS(P_j)$ allora P_i si mette in attesa (*wait*)

2) In Figura 5(a) vediamo la numerazione che si ottiene utilizzando i clock logici mentre in Figura 5(b) è rappresentato il diagramma che si ottiene nel caso dei vettori di clock logici.



(a) (b)

Figura 5: soluzione all'esercizio numero 2

3) Un client che desidera montare una directory remota invia al server sul quale si trova la directory un messaggio contenente il path remoto che vuole montare (ma non il mount point).
Il server controlla che il path corrisponda effettivamente ad un file system esportato leggendo il file `/etc/exports` e in caso affermativo ritorna al client un file handle (contenente varie informazioni tra cui il tipo di file system e il numero di i-node) che verrà utilizzato dal client per tutte le richieste successive riguardanti quella directory.

4) Esistono 4 tipi di consistenza:

Consistenza stretta: è il modello che si ha quando si ha una sola copia del dato. Ogni lettura ritorna sempre il valore memorizzato dall'ultima scrittura. Tale schema è però irrealizzabile in un ambiente distribuito.

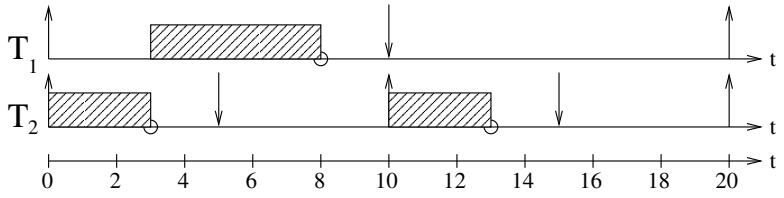
Consistenza sequenziale: i valori possono non essere letti nello stesso ordine con cui avvengono le scritture ma comunque tale ordine di lettura è uguale per tutti i processi.

Questo corrisponde a considerare corretto un qualunque scheduling dei diversi processi (anche mescolati tra loro) purché per ogni processo le operazioni vengano eseguite nell'ordine specificato dal suo programma sorgente.

Consistenza causale: in tale schema è importante soltanto l'ordine delle scritture causalmente correlate. Queste infatti sono le uniche che devono essere viste nello stesso ordine da tutti i processi, mentre le scritture concorrenti possono essere osservate in ordini diversi da processi diversi. Quindi per poter implementare tale schema è necessario tenere traccia delle dipendenze causali tra le operazioni dei diversi processi.

Consistenza debole: si utilizzano delle variabili di sincronizzazione al fine di sincronizzarsi con gli altri processi e di rendere effettive le proprie scritture.
Quando un processo si sincronizza tutte le sue scritture precedenti vengono esportate verso tutte le altre macchine (che devono comunque a loro volta sincronizzarsi per poter vedere tali modifiche) e tutte le scritture di queste ultime vengono importate.
A sua volta l'accesso alle variabili di sincronizzazione è consistente in modo sequenziale.

5) La soluzione può essere trovata disegnando il diagramma temporale dello schedule che si ottiene con RM ed osservando poi quanto le deadline possano essere anticipate al massimo rispetto al periodo pur mantenendo la schedulabilità con RM.
In questo caso, tenendo conto anche del fatto che il testo dell'esercizio impone $0{,}5 \leq \alpha < 1$, osservando la figura sottostante si può concludere che anche con α=0,5 l'insieme rimane schedulabile con RM.



Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
16 Luglio 2003

---

1. Mostrare le principali caratteristiche del meccanismo di comunicazione basato su chiamata a procedura remota (RPC), definendo in particolare la sequenza operativa ed il meccanismo di *binding*. [**4 punti**]

---

2. Descrivere i possibili *modelli di consistenza* implementabili in un sistema che utilizza la memoria condivisa distribuita (DSM), indicandone ove possibile un esempio. [**4 punti**]

---

3. Descrivere il funzionamento del meccanismo di inversione di priorità, e mostrare un esempio di come esso può portare a **situazioni di deadlock**. [**5 punti**]

---

4. Si consideri il seguente insieme di task periodici:

Task	C_i	T_i
τ_1	3	6
τ_2	3	12
τ_3	6	20

   Il sistema è in sovraccarico (cioè $U > 1$). Tuttavia, i task τ_1 e τ_2 sono in realtà task con deadline soft; in particolare, essi possono ritardare rispetto alla deadline di una unità di tempo.

   Determinare se l'insieme di task è schedulabile (soddisfa cioè deadline hard e soft, nei limiti tollerati) usando rispettivamente RM e EDF. Si mostrino i corrispondenti diagrammi temporali. [**3+2 punti**]

---

5. Si considerino i seguenti tre task periodici:

Task	T	C
τ_1	20	5
τ_2	10	3
τ_3	15	2

   Si supponga che il context switch tra processi sia non trascurabile, ed abbia una durate ϵ. Si calcoli il massimo valore di ϵ per il quale l'insieme di task risulta schedulabile usando:

   - RM (NOTA: $U(3) = 0.78$);
   - EDF.

   Il fatto che il valore di ϵ per RM sia più alto che nel caso di EDF può avere validità generale? [**5+2 punti**]

SOLUZIONE:

1) Un processo su un sistema (locale) invoca una procedura (remota) in modo che risulti trasparente all'utente, si usa quindi la lo stesso meccanismo della classica chiamata a procedura:

- richiesta: la chiamata a procedura;
- risposta: il risultato ritornato dalla procedura stessa.

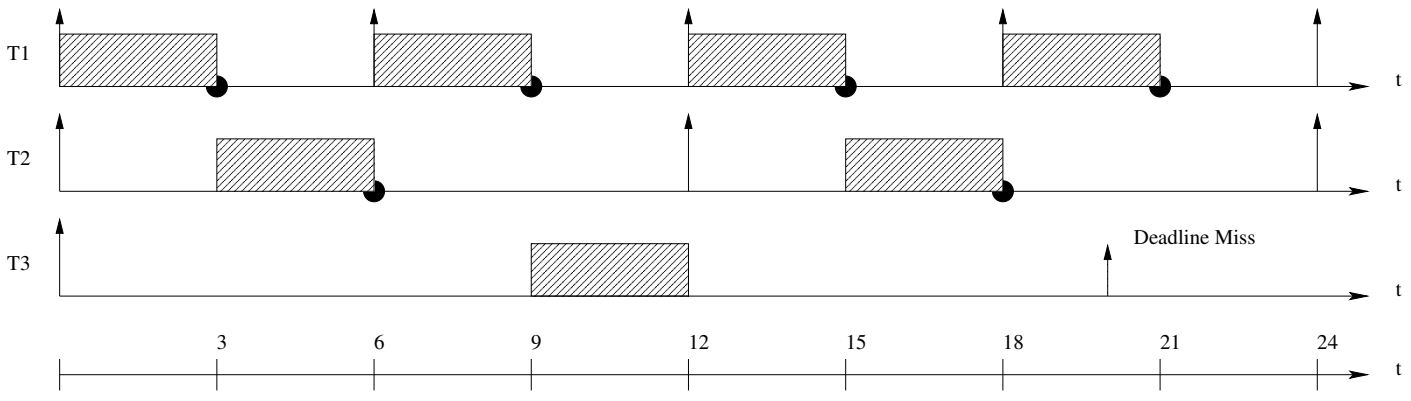
Il client invoca una procedura locale (client stub) che si preoccupa di impacchettare gli argomenti (marshaling) per la procedura remota, trasformandoli in un formato standard.
Il passo successivo la crezione dei messaggi consiste nel chiedere al kernel locale di inviare al sistema remoto ed eseguire poi una receive bloccante (per attendere il risultato).
I messaggi viaggiano sulla rete sino al sistema remoto dove vengono presi dal server stub che spacchetta gli argomenti e li converte nel formato locale. Esegue una procedura locale che invoca effettivamente la funzione del server (che il client aveva richiesto). La procedura server termina e ritorna al server stub i risultati che verranno impacchettati e inviati tramite chiamate al kernel sulla rete sino ad arrivare al client stub che li passerà al client che li aveva richiesti...
BINDING: è un meccanismo che permette al client di individuare il server.
Il server che offre un servizio RPC si registra (identificatore + porta) presso un binder (un server in esecuzione sullo stesso host del servizio RPC, in una porta nota) e il client che vuole utilizzare il servizio deve contattare il binder per ottenere l'indirizzo completo e corretto a cui la richiesta RPC deve essere inviata.

2) I meccanismi di consistenza sono necessari con la presenza di memoria distribuita perché esistono copie multiple di un valore e sorgono problemi in caso di scrittura e lettura successiva.
I modelli di consistenza visti sono 4:

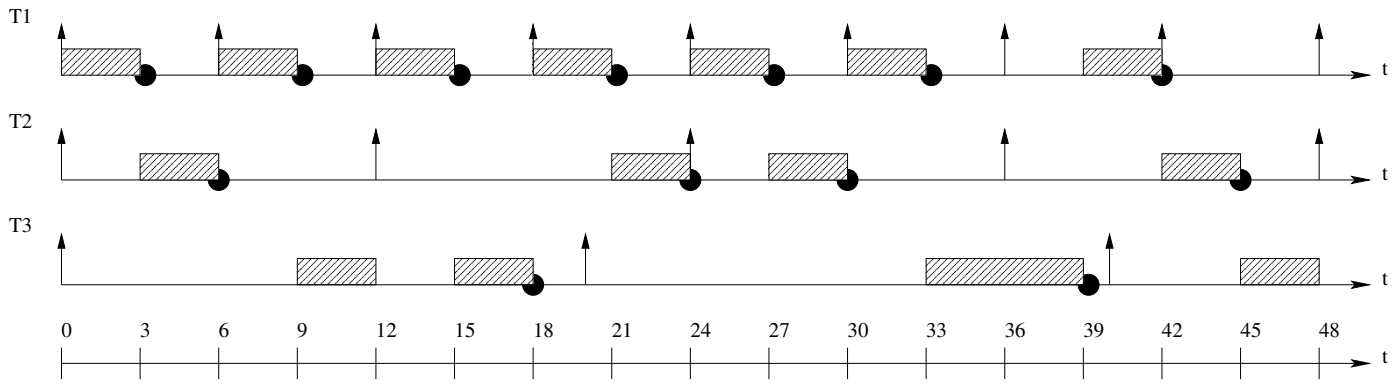
- Consistenza stretta: è il modello ideale ovvero ogni lettura di X ritorna il valore memorizzato dall'ultima scrittura di X.
- Consistenza sequenziale: il risultato di ogni esecuzione è identico al caso in cui le operazioni sono eseguite dai processi in qualche ordine sequenziale. Le operazioni di un singolo processo devono apparire nell'ordine specificato dal programma.
- Consistenza causale: scritture correlate causalmente devono essere viste nello stesso ordine da tutti i processi, mentre quelle concorrenti possono essere viste in ordini diversi da diversi processori.
- Consistenza debole: una volta sincronizzati tutti i processori hanno la stessa versione dei dati.

3) Vedi esercizio 4 del 19 Giugno 2002.

4) Utilizzando l'algoritmo RM vediamo che il task τ_3 non termina entro la sua hard deadline quindi RM fallisce:



Con l'algoritmo EDF invece vengono rispettate tutte le soft e hard deadline:



5)

a. Visto che $U(3) = 0.78$ sappiamo che per essere certi che RM sceduli correttamente l'utilizzo deve essere inferiore a U_{lub}; perciò utilizziamo proprio questa disequazione:
$U \leq 0.78 \Longrightarrow \frac{5+\varepsilon}{20} + \frac{3+\varepsilon}{10} + \frac{2+\varepsilon}{15} \leq 0.78 \Longrightarrow \varepsilon \leq 0.446$

b. Analogo al precedente è il calcolo per ottenere il valore massimo di ε, infatti basta diseguagliare U con 1 anziché 0.78, visto che EDF ci assicura una corretta scheduklazione proprio quando l'utilizzo totale è inferiore o uguale a 1: $\frac{5+\varepsilon}{20} + \frac{3+\varepsilon}{10} + \frac{2+\varepsilon}{15} \leq 1 \Longrightarrow \varepsilon \leq 1.4615$

c. Questo fatto non può avere validità generale, come dal resto si vede in questo esempio.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
10 Dicembre 2003

---

1. Si descriva un meccanismo centralizzato (basato su un coordinatore) per garantire la mutua esclusione in sistemi distribuiti. **[4 punti]**

---

2. Si definisca il modello di *consistenza sequenziale*, nel contesto di un sistema che utilizza la memoria condivisa distribuita (DSM). Si mostri poi un esempio di esecuzione con due processi, in cui si possa evidenziare la differenza tra questo modello di consistenza e la consistenza tradizionale (cioè *stretta*). **[5 punti]**

---

3. Descrivere le caratteristiche principali di NFS, mostrandone sia uno schema architetturale sia uno schema funzionale. **[4 punti]**

---

4. Siano dati due task periodici τ_1 e τ_2, con periodo rispettivamente T_1 e T_2, e con $T_1 < T_2$. Inoltre, sia detto $F = int(T_2/T_1)$ il numero di periodi di T_1 **interamente** contenuti in T_2.

   Si assuma infine che il tempo di esecuzione C_1 di τ_1 sia tale per cui $C_1 < T_2 - FT_1$, ossia che tutte le richieste di τ_1 entro il tempo T_2 siano completate.

   Si supponga quindi di applicare a questi due task l'algoritmo RM.

   Si calcolino:

   - il massimo valore di C_2 che permette di saturare l'utilizzazione della CPU;
   - usando il valore di C_2 precedentemente calcolato, si calcoli l'espressione del fattore U di utilizzazione della CPU in funzione di C_1, T_1, T_2 e F.
   - In base alle relazioni tra i valori di T_1, T_2 ed i vincoli su C_1, si disegni l'andamento di U in funzione di C_1, e si calcoli il minimo valore U_{lb} di U rispetto alla variabile C_1.

   [3+2+2 punti]

---

5. Si consideri il seguente insieme di task periodici, ed il seguente insieme di richieste (task) aperiodiche:

Task	a_i	C_i	T_i
τ_1	0	2	7
τ_2	0	1	4

Task	a_i	C_i	d_i
J_1	4	2	11
J_2	10	1	14

   dove le deadline si intendono come **assolute**. Si assuma che i task periodici vengano schedulati secondo un algoritmo RM (rate monotonic).

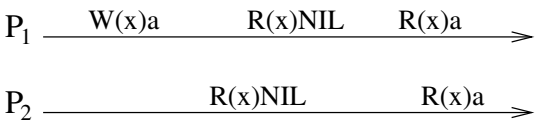
   Assumendo che le richieste aperiodiche siano gestite da un *priority server* di tipo *polling* con capacità $C_s = 2$, si calcoli il massimo periodo T_s che permette di schedulare i task rispettando tutte le deadline.

   Si assuma che il server si scarichi istantaneamente se all'inizio di un periodo non sono presenti richieste aperiodiche **[6 punti]**

SOLUZIONE:

1) Un particolare processo, detto coordinatore, si occupa di gestire l'ingresso in mutua esclusione nella sezione critica.
Un processo che desidera entrare nella SC invia un messaggio di request al coordinatore. Questo, se la sezione critica è "libera" invia immediatamente un messaggio di reply al processo consentendogli così l'ingresso. Se invece nella sezione critica è già presente un altro processo non invia il messaggio di reply ed accoda la richiesta.
Quando un processo esce dalla sezione critica lo comunica al coordinatore inviandogli un messaggio di release. Il coordinatore allora risponde alla prima richiesta accodata nel frattempo (se presente).

2) Nel modello della consistenza sequenziale il risultato di ogni possibile esecuzione (purché valida) è identico al caso in cui le operazioni siano eseguite dai processi in un qualche ordine sequenziale. In altre parole i riferimenti alla memoria sono visti nello stesso ordine da tutti i processi.
Perché un'esecuzione sia valida è necessario che le operazioni di ogni processo appaiano nell'ordine specificato dal programma del processo stesso.
La seguente figura mostra una situazione consistente dal punto di vista sequenziale dato che i processi vedono la memoria evolvere nello stesso ordine (`NIL` → `a`) ma non strettamente consistente dato che le prime due letture di P_1 e P_2 pur essendo successive alla scrittura leggono `NIL` invece di `a`.



3) NFS è basato su un'architettura client/server, anche se la distinzione tra i due è puramente concettuale dato che essi eseguono lo stesso software.
Si chiama server un host che mette a disposizione (esporta)una o pi parti del prorpio file system. I file system epsortati devono essere registrati nel file /etc/exports. Il server richiede in esecuzione 3 demoni:

- nfsd che accetta le richieste dai client e le passa a mountd;
- mountd che gestisce le richieste;
- rpcbind che consente ai client di conoscere la porta su cui è in attesa il server NFS.

I client possono integrare nel proprio file system locale i file system del server eseguendone il mount in un mount point locale. È inoltre possibile eseguire l'automounting all'avvio del client registrando opportunamente i file system nel file /etc/fstab.
Il protocollo di mount prevede che il client invii una richiesta al server contenente il path del file system a cui è interessato. Il server verifica che il path sia esportato e in caso affermativo ritorna al client un file handle che verrà utilizzato dal client per tutte le operazioni successive.
NFS prevede anche un protocollo per l'accesso a file contenente delle primitive per tutte le operazioni che possono essere eseguite sui file.
NFS è basato su RPC e di conseguenza è stateless ed indipendente dal protocollo di trasporto utilizzato dalla rete sottostante.
L'architettura NFS è divisibile in tre strati:

1. system call layer;
2. Virtual File System (VFS) layer: mantiene una tabella di v-node, uno per ogni file aperto. Ogni v-node punta ad un i-node se il file è locale o ad un r-node se è remoto;
3. NFS layer: traduce le richieste secondo il protocollo NFS.

4) a. Visto che vogliamo saturare la CPU calcoliamo l'utilizzo e lo poniamo uguale a 1:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = 1 \qquad \Rightarrow \qquad \frac{(F+1)C_1 + C_2}{T_2} = 1 \qquad \Rightarrow \qquad C_2 = T_2 - (F+1)C_1$$

in cui il primo passaggio si ha tenendo conto dei vincoli imposti dal testo dell'esercizio.

b. L'espressione di U si ottiene sostituendo il valore appena trovato nella normale espressione di U:

$$U = \frac{C_1}{T_1} + \frac{T_2 - (F+1)C_1}{T_2}$$

5) L'esercizio riguarda argomenti non pi trattati nel corso.

Università degli studi di Verona
Corso di Laurea in Informatica/Tecnologie dell'Informazione
Sistemi Operativi Avanzati
30 Marzo 2004

---

1. Si descriva un meccanismo centralizzato (basato su un coordinatore) per garantire la mutua esclusione in sistemi distribuiti.
   [**5 punti**]

---

2. Si consideri il seguente insieme di task a, b, c, d, che devono accedere a due risorse condivise (semafori) P e Q. I task hanno le seguenti caratteristiche:

Task	*Priorità*	*Tempo di rilascio*	*Sequenza di operazioni*
a	4	4	EEQPE
b	3	2	EPPE
c	2	2	EE
d	1	0	EQQQQE

   Le operazioni possibili sono 3: E (esecuzione), P (accesso a P) e Q (accesso a Q). Ogni istanza di operazione indica un'unità di tempo. Per esempio, EVVE = (E per un'unità di tempo, Q per due unità di tempo, etc.).

   Mostrare l'esecuzione dei 4 processi nei tre casi:

   (a) Utilizzo di un normale algoritmo a priorità (4=max, 1=min);

   (b) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando la *priority inheritance*;

   (c) Utilizzo di un normale algoritmo a priorità e gestione dei semafori usando il *priority ceiling*;

   Si mostrino i diagrammi temporali, e si calcoli per ogni task il tempo di risposta.
   [**8 punti**]

---

3. Si valuti la schedulabilità del seguente insieme di task periodici:

Task	a_i	C_i	d_i
J_1	0	2	9
J_2	1	3	11
J_3	0	1	4
J_4	3	2	8
J_4	2	1	5

   [**5 punti**]

---

4. Costruire un insieme di quattro task periodici che sia schedulabile con RM, e per il quale il test di utilizzazione basato su U fallisca.
   NOTA: $U(4) \approx 0.757$.
   [**4 punti**]

---

5. Mostrare un esempio di gestione della memoria condivisa distribuita (DSM) che non faccia uso di replicazione dei dati.
   [**4 punti**]

---

6. Si descriva il funzionamento di un *polling server*, illustrandone il principio di base.
   [**4 punti**]

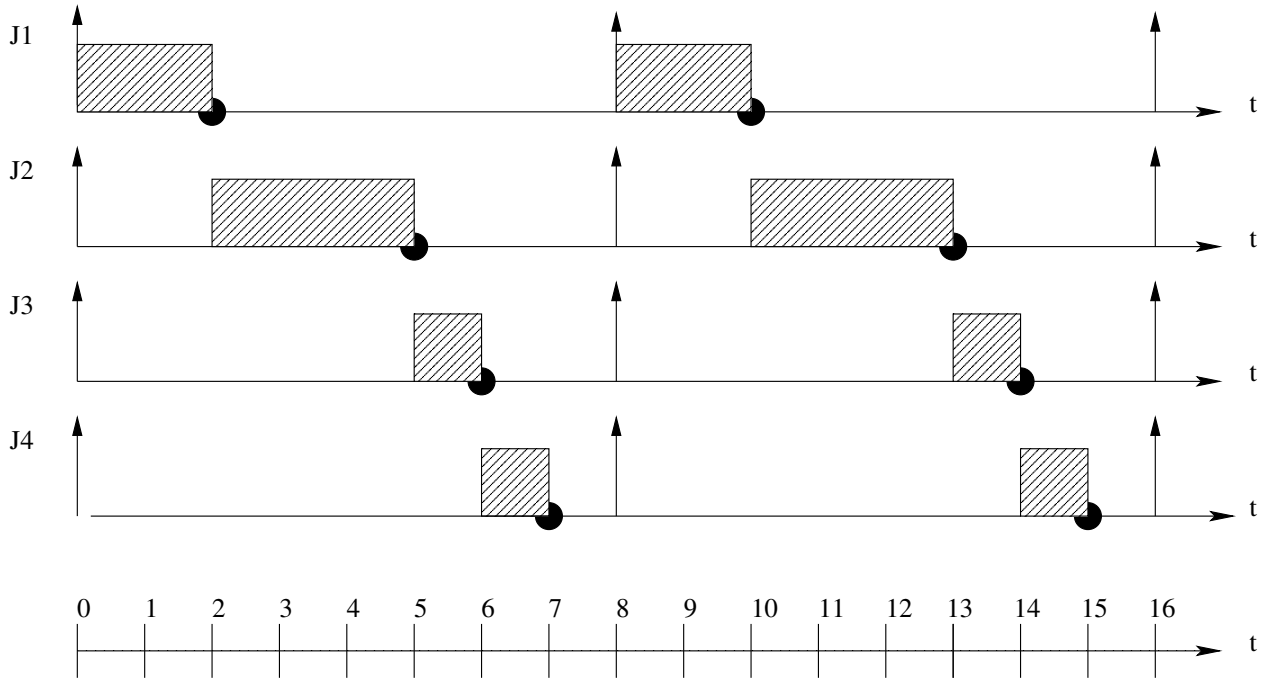
SOLUZIONE:

1) Vedi esercizio 1 del 10 Dicembre 2003

2) Vedi esercizio 2 del 03 Dicembre 2002

3) L'insieme di task periodici risulta essere non schedulabile in quanto provando a fare l'analisi sul tempo di risposta si osserva che sia J_5 che J_3 hanno un tempo di risposta superiore alle loro deadline. Questa analisi fornisce una condizione sufficiente e necessaria per la schedulabilità quindi concludiamo che l'insieme è non schedulabile.

4) L'insieme di task proposti presenta un utilizzo totale di 0.875, ma da come si può osservare dal diagramma temporale, esso è perfettamente schedulabile:



5) Tra gli approcci di DSM visti, 2 riguardano la non replicazione dei dati:

- Central Server
- Data Migration

Parlando del *Central Server* si deve ricordare che esiste un server centrale appunto che mantiene tutte le pagine dati e a seconda dei casi ritorna i dati in caso di lettura oppure effettua le modifiche in caso di scrittura.
Tra i vantaggi ricordiamo la facile implementazione e i meccanismi di sincronizzazione consolidati; tra gli svantaggi invece bisogna sottolineare che la maggior parte dei riferimenti a dati sono remoti e quindi è soggetto a colli di bottiglia che esso stesso rappresenta in caso di molteplici richieste.

6) Il *polling server* è un priority server STATICO. Il server ha anch'esso una priorità associata al suo periodo di esecuzione, inoltre è caratterizzato da una capacità
Il principio di funzionamento è questo: i task periodici vengono schedulati normalmente con RM quando è il turno del server questo può servire gli eventi aperiodici (tanti quanti è consentito dala capacità, se ce ne sono, altrimenti si scarica e si ricaricherà all'inizio del successivo periodo.

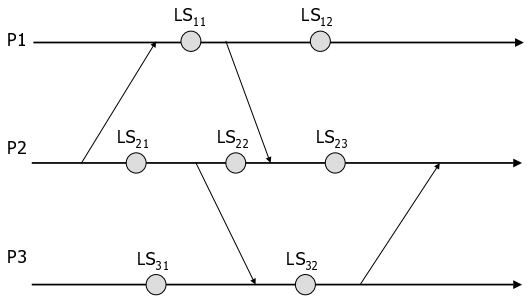
Università degli studi di Verona
Corso di Laurea Specialistica in Informatica
Sistemi Operativi Avanzati
22 Giugno 2004

---

1. Descrivere il funzionamento del meccanismo di inversione di priorità, e mostrare un esempio di come esso può portare a situazioni di deadlock. **[6 punti]**

---

2. Si consideri l'insieme di tre processi P_1, P_2 e P_3 indicato in figura, le frecce indicano l'invio di un messaggio, ed i cerchi uno stato *locale* di ogni processo. Per ogni possibile stato globale costituito da una terna di stati locali $GS = (LS_{1i}, LS_{2j}, LS_{3k}), \forall i, j, k$, si determini se è uno stato consistente, e, in caso contrario si indichi il perchè.



NOTA: Ci sono $12 = 2 \cdot 3 \cdot 3$ possibili stati globali. **[7 punti]**

---

3. Si consideri il seguente insieme di task periodici:

Task	T	C
1	100	30
2	50	10

Assumendo che la durata di un context switch sia non nulla, si calcoli la massima durata possibile del context switch tale per cui l'insieme dei due task sia schedulabile, usando rispettivamente:

(a) EDF

(b) RM

E' possibile che tale valore sia più grande per RM che non per EDF?

NOTA: Per context switch si intenda l'intero tempo richiesto per passare dall'esecuzione di un task ad un altro.

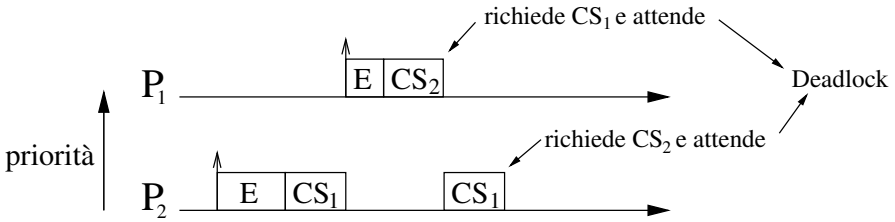
[3+3+2 punti]

---

4. Si descriva il principio di funzionamento di uno schema di memoria condivisa distribuita (DSM) basato su *read replication*. Si descriva in particolare, il comportamento di un client e dell'host remoto nei casi di scrittura e lettura di un blocco (pagina).
[4 punti]

SOLUZIONE:

1) Si ha inversione di priorità quando un processo ad alta priorità è costretto a sospendere la propria esecuzione a tempo indefinito in attesa che un proceso a pi bassa priorità liberi una sezione critica in comune ad entrambi.
Il caso pi comune si ha quando P_2 entra in una sezione critica, viene sospeso per poter eseguire P_1 ($Prio(P_1)>Prio(P_2)$) ma questo a sua volta si blocca perché ha bisogno di entrare nella sezione critica ancora occupata da P_2. A questo punto P_2 torna in esecuzione e P_1 deve attendere. Il problema maggiore risulta però dal fatto che P_2 a sua volta può ora essere sospeso per eseguire altri processi a pi alta priorità aumentando così il tempo per cui P_1 rimane bloccato.
L'inversione di priorità può anche portare a deadlock. Consideriamo infatti il caso in cui P_1 (ad alta priorità necessiti di due sezioni critiche: CS_2 e CS_1 (innestata in CS_2), mentre P_2 (a bassa priorità necessiti di CS_1 e CS_2 (innestata in CS_1). Allora è possibile che i due processi vadano in deadlock come mostrato nella figura sottostante.



2) Uno stato si dice consistente se per ogni evento che include, esso include anche tutti gli eventi che lo precedono causalmente. I 12 stati globali possibili possono allora essere così classificati:

- (LS_{11}, LS_{21}, LS_{31}) è consistente
- (LS_{11}, LS_{21}, LS_{32}) non è consistente perché include (R, m_2) ma non (S, m_2)
- (LS_{11}, LS_{22}, LS_{31}) è consistente
- (LS_{11}, LS_{22}, LS_{32}) è consistente
- (LS_{11}, LS_{23}, LS_{31}) non è consistente perché include (R, m_3) ma non (S, m_3)
- (LS_{11}, LS_{23}, LS_{32}) non è consistente perché include (R, m_3) ma non (S, m_3)
- (LS_{12}, LS_{21}, LS_{31}) è consistente
- (LS_{12}, LS_{21}, LS_{32}) non è consistente perché include (R, m_2) ma non (S, m_2)
- (LS_{12}, LS_{22}, LS_{31}) è consistente
- (LS_{12}, LS_{22}, LS_{32}) è consistente
- (LS_{12}, LS_{23}, LS_{31}) è consistente
- (LS_{12}, LS_{23}, LS_{32}) è consistente

3) Per risolvere l'esercizio possiamo considerare il context switch come un tempo fisso che viene aggiunto ai tempi d'esecuzione dei due processi.
Quindi se indichiamo con *cs* il tempo per un context switch i tempi d'esecuzione diventano: $C_1 = 30 + cs$ e $C_2 = 10 + cs$.
a. Perché l'insieme dei task sia schedulabile con EDF è necessario che $U \leq 1$ dunque calcoliamo di conseguenza il valore di *cs*.

$$U = \frac{30+cs}{100} + \frac{10+cs}{50} = \frac{1}{2} + \frac{3cs}{100} \leq 1$$

da cui

$$cs \leq \frac{50}{3} = 16,67 \approx 16$$

In realtà questo valore considera un context switch anche tra esecuzioni successive dello stesso processo. Ed in effetti utilizzando EDF agli istanti t*100 riparte per la seconda volta successiva T_2 e quindi non è necessario considerare un context switch.
Per cui il valore effettivo di cs può essere calcolato come $16 + \frac{18}{2} = 25$, dove 18 è il tempo che, considerando cs=16, rimane libero tra un'esecuzione di T_2 e la successiva (come mostra la Figura 6).

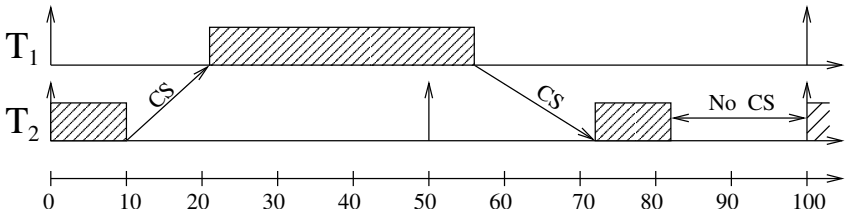


Figura 6:

b. utilizzando la medesima idea del punto a. e il test di schedulabilità basato sul fattore di utilizzo si ottiene:

$$U \;=\; \frac{30+cs}{100} \;+\; \frac{10+cs}{50} \;=\; \frac{1}{2} \;+\; \frac{3cs}{100} \;\leq\; 0,828$$

da cui

$$cs \;\leq\; 10,93 \;\approx\; 10$$

In realtà questo tipo di test rappresenta una condizione sufficiente ma non necessaria ed infatti l'insieme dei task è schedulabile con RM anche con cs=12.

4) Lo schema per la gestione della memoria condivisa distribuita basato su read replication implementa un protocollo di tipo "lettore multiplo / scrittore singolo".
Ogni pagina ha un possessore che mantiene sempre la versione pi aggiornata e memorizza la pagina in modo permanente.
In caso di lettura:

1. il client fa richiesta della pagina al rispettivo possessore;
2. l'host remoto una volta ricevuta la richiesta imposta come read-only la pagina in modo che non possa essere modificata da un altro host durante il suo trasferimento. In questo modo il client non riceve mai una versione già vecchia;
3. il client riceve la pagina, la imposta come in sola lettura ed invia un messaggio di acknowledgement all'host remoto;
4. la macchina remota una volta ricevuto l'acknowledge modifica i permessi della propria copia della pagina in lettura/scrittura.

In caso di scrittura invece:

1. il client prima di tutto se deve scrivere su una pagina di cui non possiede una copia ne fa richiesta al possessore utilizzando il protocollo per la lettura. Quando possiede una copia della pagina invia al possessore la richiesta di scrittura;
2. il possessore invalida tutte le copie;
3. il client a questo punto accede in scrittura alla pagina e poi invia la pagina modificata al possessore che aggiorna la propria copia.

Naturalmente perché sia possibile invalidare tutte le copie di una pagina è necessario tenere traccia di queste copie (o se ne occupa il possessore o si utilizza una lista distribuita).
Questo schema è efficiente se il numero di read è molto superiore al numero di write, inoltre dato che le letture sono normalmente ad alta località la maggior parte di queste saranno locali (dopo la prima). Tra gli svantaggi di questo approccio si ricordano la necessità di invalidare le diverse copie di una pagina in scrittura e la possibilità di un sovraccarico di lavoro per i possessori di pagine molto richieste.

Università degli studi di Verona
Corso di Laurea Specialistica in Informatica
Sistemi Operativi Avanzati
8 Luglio 2004

---

1. Mostrare la sequenza di messaggi risultante dall'applicazione dell'algoritmo di elezione *bully*, nel caso di 6 $(P_0, \ldots, P_5)$ processi, in seguito alla seguente sequenza di eventi:

   (a) L'attuale coordinatore P_5 va in crash;

   (b) Il processo P_3 lancia per primo un'elezione.

   (c) Dopo che il nuovo coordinatore è stato eletto, il processo P_6 si riattiva.

   Si descriva brevemente il principio usato dall'algoritmo.

   [**1+1+1+2 punti**]

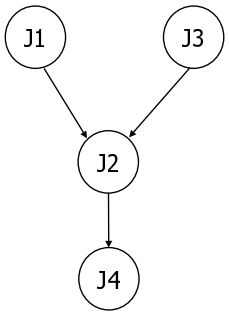
---

2. Si descriva il meccanismo di funzionamento dello schema di comunicazione basato su chiamate di procedura remota (RPC)

   [**4 punti**]

---

3. Sia dato il seguente insieme di task aperiodici, legati dal seguente grafo delle dipendenze:



Task	C_i	d_i
J_1	3	5
J_2	2	6
J_3	1	3
J_4	3	9

   Fornire una schedulazione valida dei task utilizzando l'algoritmo LDF.

   [**5 punti**]

---

4. Si consideri un *polling server* in cui sia il server che gli n task periodici siano gestiti secondo l'algoritmo RM.

   Si assuma poi che il server, avente capacità C_s e periodo T_s, venga attivato ad ogni periodo e consumi sempre tutta la sua capacità in ogni periodo.

   Si discuta la schedulabilità dell'insieme dei task, indicando una formula funzione dei parametri sopra indicati. Si indichi con $(C_i, T_i), i = 1, \ldots, n$ il tempo di esecuzione ed il periodo di ciascun task periodico.

   [**7 punti**]

---

5. Si costruisca uno schedule **statico** per il seguente insieme di task:

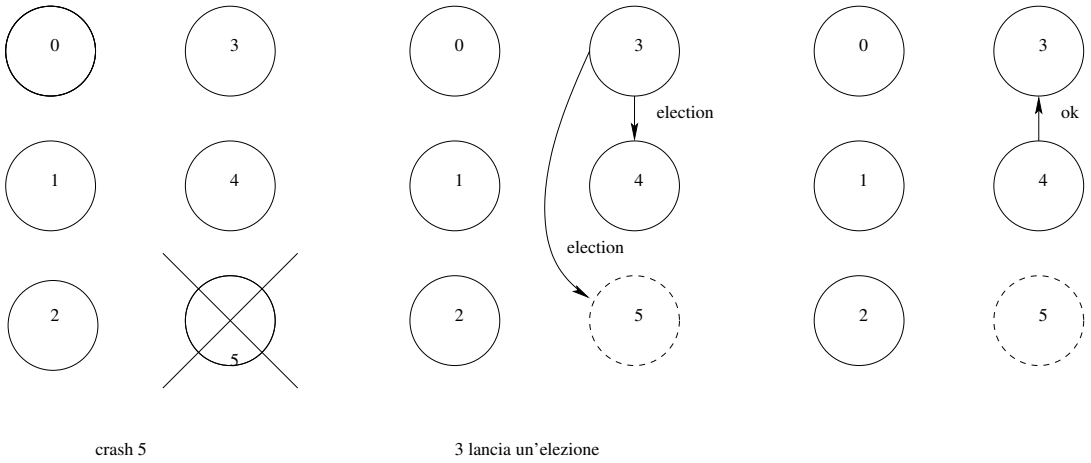
Task	T_i	C_i
τ_1	4	1
τ_2	5	1.8
τ_3	20	1
τ_4	20	2

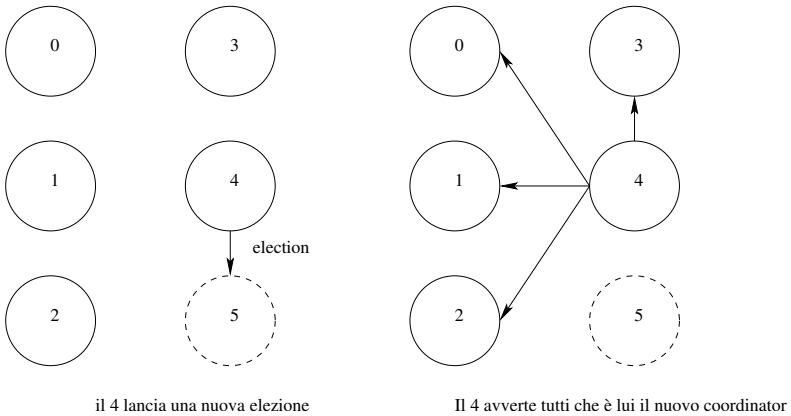
   Si indichino chiaramente minor e major cycle

   [**5 punti**]

SOLUZIONE:

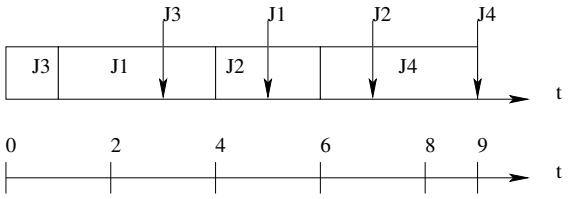
1) Nella figura viene mostrata la sequenza a) e b). Una volta che P_5 si riattiva potrebbe rilanciare una nuova elezione e diventare così il nuovo coordinator oppure non fare niente.



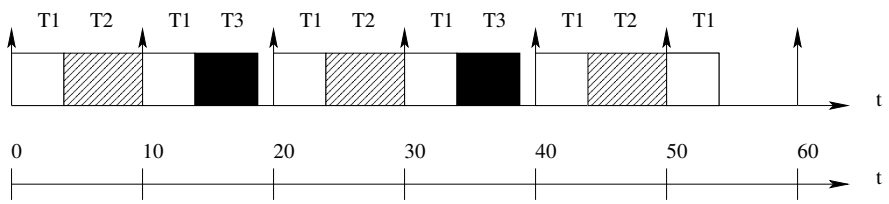


2) In sintesi il meccanismo RPC prevede che un processo su un sistema locale invochi una procedura remota in modo trasparente. Per questo meccanismo sono necessari gli stub (client e server) che si occupano della serializzazione e deserializzazione della richiesta e della risposta, in pi si occupano di inoltrare la richiesta e la risposta al processo client e al processo server.

3) Con LDF, dato l'albero delle precedenze, si prende tra le foglie il processo che ha deadline maggiore e si mette in coda. Si continua sino a che non ci sono pi processi da schedulare. Con questo algoritmo si ottiene il seguente digramma temporale:



5) Banalmente si calcola che il minor cycle è 10 e il major cycle è 60:



Università degli studi di Verona
Corso di Laurea in Informatica/Sistemi Intelligenti e Multimediali

Sistemi Operativi Avanzati
23 Marzo 2005

Nome	Congnome	Matricola	Laurea	Master

Esercizi

1. Si consideri di utilizzare l'algoritmo RM per schedulare il seguente insieme di task periodici:

$Task$	τ_1	τ_2	τ_3
C_i	1	2	3
T_i	4	6	10

   (a) Verificare la schedulabilità eseguendo un'analisi basata sul processor utilization factor.

   (b) Verificare la schedulabilità eseguendo un'analisi basata sul worst case response time.

   (c) Costruire il diagramma dello schedule.

[2+4+2 punti]

2. Si consideri un sistema distribuito con memoria condivisa distribuita.

   (a) Si descriva cosa si intende con il concetto di consistenza.

   (b) Si descrivano i modelli di consistenza stretta, sequenziale, causale, debole.

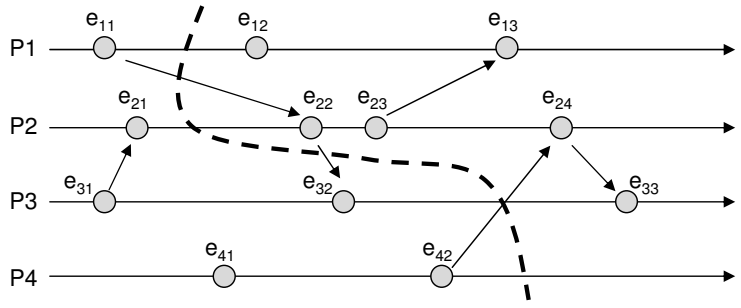
   (c) Si mostri un esempio in cui un insieme di processi sono consistenti causalmente, ma non sequenzialmente.

[1+3+3 punti]

3. Si consideri il problema della sincronizzazione dei clock in un sistema distribuito.

   (a) Si descriva cosa si intende per violazione della causalità quando si utilizza il metodo di sincronizzazione di Lamport basato sul concetto di clock logico.

   (b) Utilizzare l'algoritmo di Fidge/Matten (vettori di clock) per sincronizzare il seguente insieme di processi. Etichettare i nodi direttamente su questo foglio.



   (c) Il taglio indicato dalla line tratteggiata nella figura precedente è consistente? È transitless? Motivare le risposte. Disegnare un taglio che sia consitente e transitless.

[2+3+2 punti]

4. Si descrivano in generale i concetti di allocazione statica e allocazione dinamica di processi in un sistema distribuito (non gli algoritmi) evidenziando in particolare motivazioni e problematiche di progetto. Quindi, si descriva il funzionamento e la struttura di Condor. [**6 punti**]

5. Barrare con una croce le risposte esatte direttamente su questo foglio. Per ogni domanda può esserci più di una risposta esatta. Si tenga presente che ogni risposta corretta vale 1 punto, mentre ogni errore vale -1.

   - Si consideri un sistema distribuito. Quali delle seguenti affermazioni sono false?
     (a) Le CPU possono eseguire sistemi operativi diversi.
     (b) La comunicazione tra i vari host è possibile solo usando memoria condivisa.
     (c) Tutti i sistemi distribuiti sono MIMD.
     (d) SMP e Cluster sono entrambi sistemi distribuiti.
   - Quale algoritmo di scheduling real time è ottimo per risolvere il problema $1|prec, sync|L_{max}$?
     (a) Algoritmo di Horn
     (b) Algoritmo di Jackson
     (c) LDF
     (d) EDF*
   - Si consideri un sistema con file system distribuito. Quali delle seguenti affermazioni sono false?
     (a) NFS è stateful.
     (b) Il meccanismo di consistenza dei file *write on close* prevede che dopo ogni scrittura il file venga chiuso per garantire l'aggiornamento della copia del server.
     (c) La caratteristica più importante del concetto di transazione atomica è la serializzazione.
     (d) Un file server stateless è maggiormente tollerante ai guasti rispetto ad un file server stateful.

N.B. Non sono ammesse domande al docente. Scrivere nome, cognome e matricola su tutti i fogli. Non consegnare la brutta copia. Soluzioni multiple discordanti dello stesso esercizio verranno valutate con punti 0.

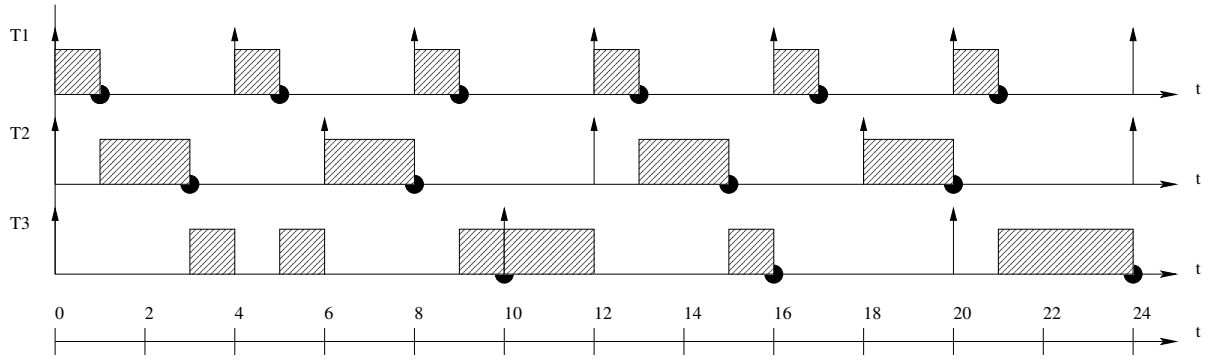
SOLUZIONE:

1)

a. Considerando l'utilizzo totale risulta non soddisfatta la condizione di $U \leq U_{lub} \Longrightarrow \frac{1}{4} + \frac{2}{6} + \frac{3}{10} > U(3)$ perciò al momento non si può dire nulla.

b. Considerando invece l'analisi basata sul worst case response time si conclude che l'insieme di task è schedulabile in quanto ogni processo rispetta la condizione $R_i \leq D_i$:
$\omega_1^0 = 1 \Longrightarrow 1 \leq 4\bullet$
$\omega_2^0 = 2$
$\omega_2^1 = 2 + \lceil \frac{2}{4} \rceil 1 = 3$
$\omega_2^2 = 2 + \lceil \frac{3}{4} \rceil 1 = 3 \Longrightarrow 3 \leq 6\bullet$
$\omega_3^0 = 3$
$\omega_3^1 = 3 + \lceil \frac{3}{4} \rceil 1 + \lceil \frac{3}{6} \rceil 2 = 6$
$\omega_3^2 = 3 + \lceil \frac{6}{4} \rceil 1 + \lceil \frac{6}{6} \rceil 2 = 7$
$\omega_3^3 = 3 + \lceil \frac{7}{4} \rceil 1 + \lceil \frac{7}{6} \rceil 2 = 9$
$\omega_3^4 = 3 + \lceil \frac{9}{4} \rceil 1 + \lceil \frac{9}{6} \rceil 2 = 10$
$\omega_3^5 = 3 + \lceil \frac{10}{4} \rceil 1 + \lceil \frac{10}{6} \rceil 2 = 10 \Longrightarrow 10 \leq 10\bullet$

c. Il diagramma temporale risultante dopo aver applicato RM è il seguente:



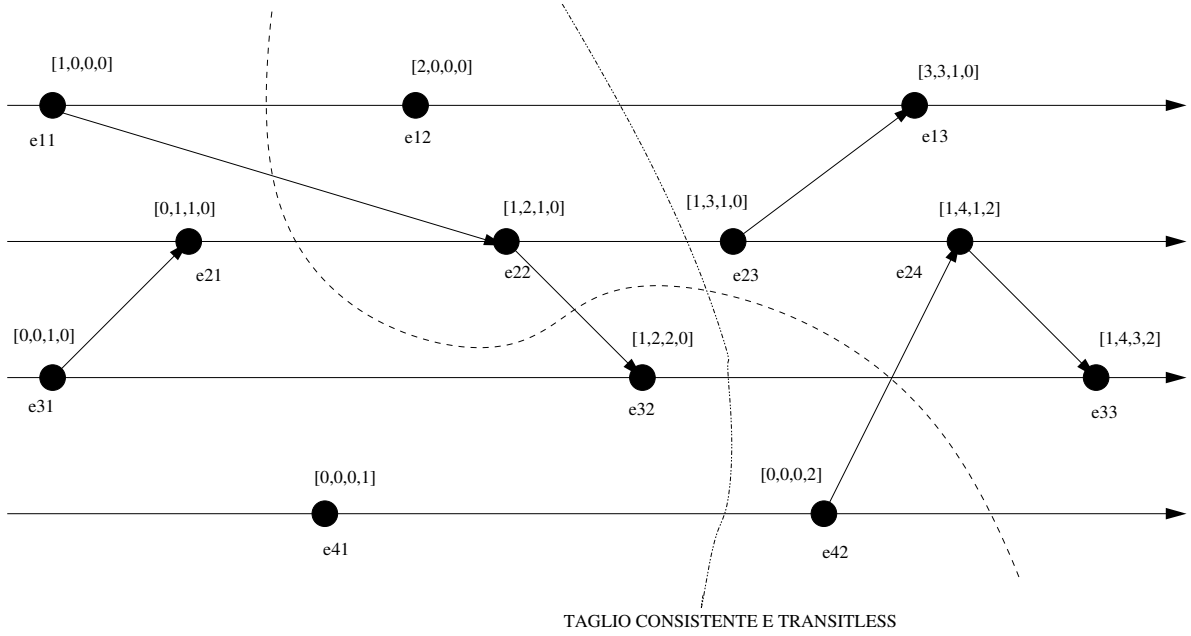
2) Vedi esercizio 2 del 16 Luglio 2003.

3)

a. Per violazione della causalità si intende la limitazione di $\rightarrow$, ovvero che se C(a)< C(b) non sappiamo se a $\rightarrow$ b.

c. Il primo taglio non è ne consistente (perché e_{32} riceve un messaggio che non è partito), ne transitless (perché non tutte le send sono state ricevute)

b.



4) Per allocazione statica (non migratoria) si intende che l'allocazione di un processo viene fatta all'atto della sua creazione una volta creato un processo non può essere pi spostato (non-preemptive).
Per migrazione (allocazione dinamica) si intende invece che un processo può essere spostato in qualunque momento della sua esecuzione (preemptive).
Le problematiche di progetto pi importanti sono relative al tipo di decisione (algoritmo) da prendere: centralizzato vs distribuito, locale vs globale, iniziato da mittente/destinatario, deterministico vs non deterministico, ottimo vs sub-ottimo.

5) Risposte da segnare:

- a,b
- c
- a,b