Prima provetta di Sistemi Operativi per la Robotica - 23/4/2009

Soluzioni

1 (2pt) Quali estensioni alla schedulazione tradizionale introduce lo standard POSIX 1003.1b?

Soluzione

le principali estensioni riguardano:

- La schedulazione

Posix1b introduce tre modalità di schedulazione:

- SCHED_FIFO: basata sulla priorita
- SCHED_RR: basata sul quanto
- SCHED_OTHER: definibile dall'utente.

- La comunicazione tra processi

Posix1b introduce tre modalità di IPC:

- Messaggi
- Memoria condivisa

- Sincronizzazione tra processi
  - Semafori
  - Mutex e variabili condivise
- I segnali

  Posix1b introduce i segnali real-time

---

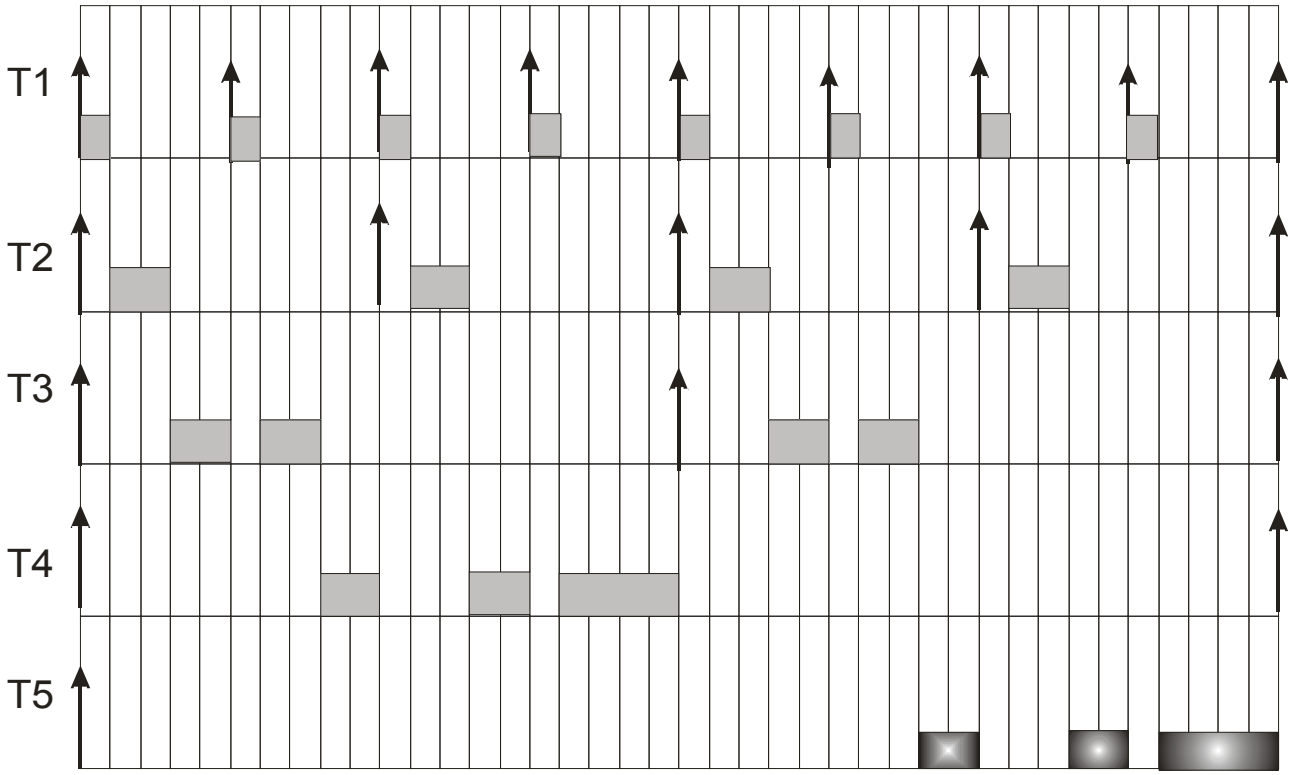
2 (4pt) Un sistema fisico è controllato da un Sistema Operativo in tempo reale che gestisce i task in tempo reale con la schedulazione Rate Monotonic. Il sistema è descritto da cinque grandezze fisiche che devono essere campionate rispettivamente ogni 5, 10, 20, 40 e 5000 microsecondi. Le durate massime dei 5 task sono pari a 1, 2, 4, 8 e 100 microsecondi.
Verificare se il sistema di task periodici è schedulabile.

Soluzione

Il calcolo del fattore di utilizzazione fornisce;
U=1/5+2/10+4/20+8/40+100/5000=0.82 che è maggiore ddi Ulsm che per 5 task è circa 0.74. E' però minore di 1 e quindi è il tipico caso che si può verificare con il metodo del tempo di risposta massimo.
Il metodo delle risposte massime fornisce per i primi 4 task:
R1=C1
R2=C2+ceiling(R2/T1)*C1
R3=C3+ceiling(R3/T1)*C1+ceiling(R3/T2)*C2
R4=C4+ceiling(R4/T1)*C1+ceiling(R4/T2)*C2+ceiling(R4/T3)*C3

che fornisce i seguenti valori (si trascurano i passaggi): R1=1; R2=3; R3=8; R4=20 che sono tutti all'interno delle loro deadline. Per R5 si dovrebbe eseguire il seguente calcolo:
R5=C5+ceiling(R5/T1)*C1+ceiling(R5/T2)*C2+ceiling(R5/T3)*C3+ceiling(R5/T4)*C4

La soluzione di quest'ultima espressione si puo' risolvere col solito metodo della convergenza dei risultati. Tuttavia questa soluzione porta a una condizione di convergenza molto lunga se fatta a mano. Sarebbe ovviamente semplice risolvere il calcolo con un programma.
Proviamo allora questa strada. I primi 4 task terminano all'interno delle loro deadline. Vediamo qual'è il tempo totale disponibile all'interno dell'iperperiodo dei primi 4 task. Tale quantità può essere calcolata; tuttavia è semplice una visualizzazione grafica.
Una rappresentazione grafica della schedulazione all'interno dell'iperperiodo dei primi 4 task è ovviamente la seguente:



(il tempo libero è indicato con il riempimento non uniforme)

dove si vede che il tempo libero nell'iperperiodo è 8 unità temporali. Un iperperiodo è di 40 unità temporali; nel periodo di 5000 ci sono 125 iperperiodi; quindi nel periodo di 5000 c'e' tempo disponibile pari a 125x8=1000 unità temporali.Visto che la durata del 5o task è di 100 unità temporali c'e' tutto il tempo per eseguire tutti i task e quindi il sistema di task è schedulabile.

---

3 (2pt) Definire l'ottimalità di una schedulazione e dire se i seguenti algoritmi di schedulazione per task aperiodici:

Horn Bratley LDF in_background

e i seguenti algoritmi per task periodici:

Rate Monotonic EDF

sono ottimali o no.

Soluzione

L'ottimalità di una schedulazione Real-Time è definita come segue:

> *Se un insieme di task aperiodici non è schedulabile con un particolare algoritmo RT, allora non è schedulabile con nessun altro algoritmo*

In pratica, conviene per la dimostrazione usare la seguente definizione equivalente:

> *Se un insieme di task è schedulabile con un qualche algoritmo A, allora sicuramente è schedulabile con l'algoritmo RT*

Ciò detto, anche se non si è parlato a lezione di ottimalità di Bratley o LDF, una semplice riflessione porta a quanto segue:

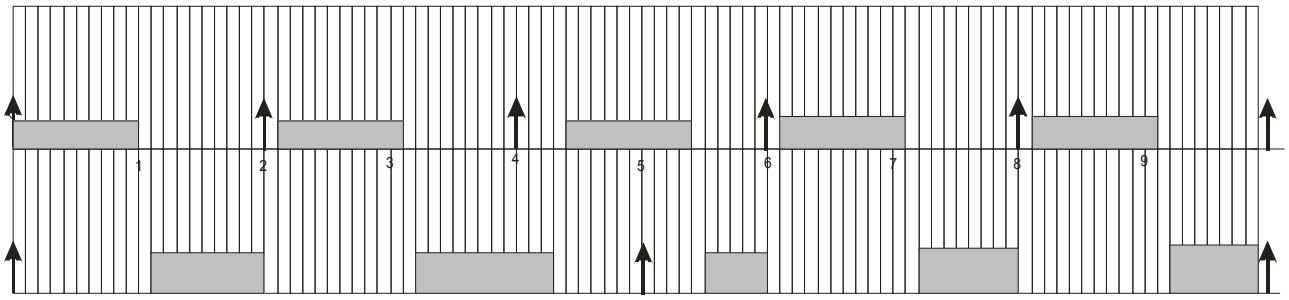
Horn	ottimo	si dimostra che se i task sono schedulabili con qualsiasi algoritmo di scheduazione è possibile applicare una trasformazione che schedula i task con Horn(EDF aperiodico) tale che Horn è fattibile
Bratley	ottimo	Bratley prova tutte le possibilità: se i task sono

		schedulabili allora Bratley trova la migliore schedulazione
LDF	ottimo	LDF minimizza la lateness massima, e se i task sono schedulabili con i vincoli di precedenza con qualsiasi algoritmo, LDF non può far altro che migliorare le cose
in_background	non ottimo	se un insieme di task è schedulabile non è detto che lo sia con la schedulazione in background che pone ulteriori vincoli
Rate Monotonic	ottimo	visto a lezione
EDF	ottimo	visto a lezione

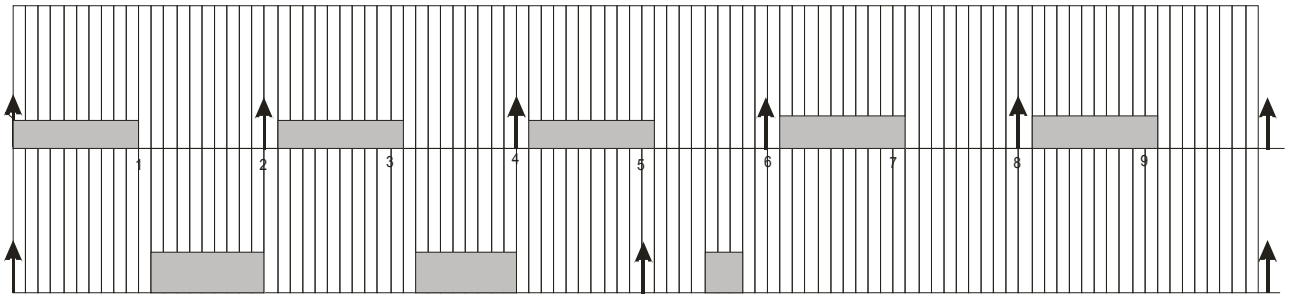
4 (5pt) In un sistema ci sono due task periodici, T1 e T2, con periodo rispettivamente pari a 2 e 5 secondi e durata pari a 1 e 2 secondi. Rispondere alle seguenti domande:
4a. Se il context switch è istantaneo, il sistema e' schedulabile con EDF?
4b. Se il context switch è di 0.1 unità temporali, è possibile schedulare il sistema con EDF?
4c. Se il context switch è di 0.1 unità temporali, è possibile schedulare il sistema con Rate Monotonic?

Soluzione

Per vedere se il sistema è schedulabile con EDF devo calcolare il fattore di utilizzazione:
U=1/2+2/5=0.9 quindi essenso minore di 1 il sistema è schedulabile con EDF.
Introduciamo il tempo di context switch. Chiariamo le condizioni: il primo task arriva periodicamente agli istanti 0, 2, 4, 6 … mentre il secondo agli istanti 5, 10, 15 …; quando un task finisce, prima di attivare l'altro aspetto 0.1 unità temporali.
Con EDF viene schedulato di volta in volta il task con deadline più imminente.
Osserviamo il grafico di schedulazione con EDF fino all'iperperiodo (ogni riga verticale rappresenta 0.1 unità temporali):



All'istante 0 ci sono entrambi i task ma viene schedulato T1 perche' ha la deadline più imminente. Il secondo task esegue fino all'istante 2, quando arriva il task 1 che fa preemption sul task 2 perchè ha la deadline più imminente. Il task 1 esegue fino all'istante 3.1. A quel punto riprende il task 2 fino all'istante 4 quando arriva il task 1. A questo punto però il task 2 ha la deadline più imminente e continua l'esecuzione. In conclusione, EDF riesce a schedulare tutti i task entro la deadline. Dopo l'iperperiodo le cose si ripetono.
Consideriamo ora la schedulazione RM (grafico seguente):



In questo caso il task 1 viene sempre schedulato all'inizio del suo periodo perchè ha la priorità più elevata. Il task 2 esegue a minore priorità. E' visualizzato solo l'esecuzione di 1 periodo di task 2. Si vede che il task 2 eccede la sua deadline e quindi il sistema non è schedulabile con RM.

---

5 (6pt) In un sistema dedicato ci sono tre task con le seguenti caratteristiche

	periodo T	durata C	priorità
T1	2	0.4	alta
T2	3	1	media
T3	5	1	bassa

Rispondere alle seguenti domande

5a. Il sistema di task e' schedulabile con un algoritmo a priorità fissa?

5b. Qual'e' il più piccolo periodo di T1 tale che il sistema sia schedulabile con EDF?

Soluzione

Prendiamo come riferimento di un algoritmo a priorita' fissa il Rate Monotonic.
Vediamo dunque se il sistema di task è schedulabile con RM.
Fattore di utilizzazione: U=0.4/2+1/3+1/5=0.73. Quindi il sistema è schedulabile con RM, visto che Ulsm=0.82, quindi è schedulabile con un algoritmo a priorità fissa.

Per calcolare il più piccolo valore di T1 basta risolvere la seguente disequazione:
0.4/T1+1/3+1/5 < 1 che fornisce T1 > 0.869 che è il valore più piccolo che garantisce la schedulazione con EDF.

---

6 (2.5pt) Qual'è l'uscita di questi due programmi POSIX?

```
ProgA()
{
        int i=0;
        printf("start\n");
        fork():
        printf("end %d\n", i++);
}
```

```
ProgB()
{
        int i=0;
        printf("start\n");
        vfork():
        printf("end %d\n", i++);
}
```

Soluzione

vfork ha lo stesso effetto di fork (crezione di un processo), senza duplicare lo spazio di indirizzamento. La memoria del padre viene quindi condivisa. La variabile i viene quindi condivisa tra i due processi. Chi esegue l'incremento di i (i++)? se il padre è più veloce allora è il padre, che stampa 0 e incrementa i. Se il figlio è più veloce è il figlio, che stampa i e la incrementa.
Il programma è un caso tipico di nondeterminatezza. I risultati dei due programmi sono:

ProgA:	ProgB
start	start
end 0	end 0
end 0	end ?? (indefinito)

---

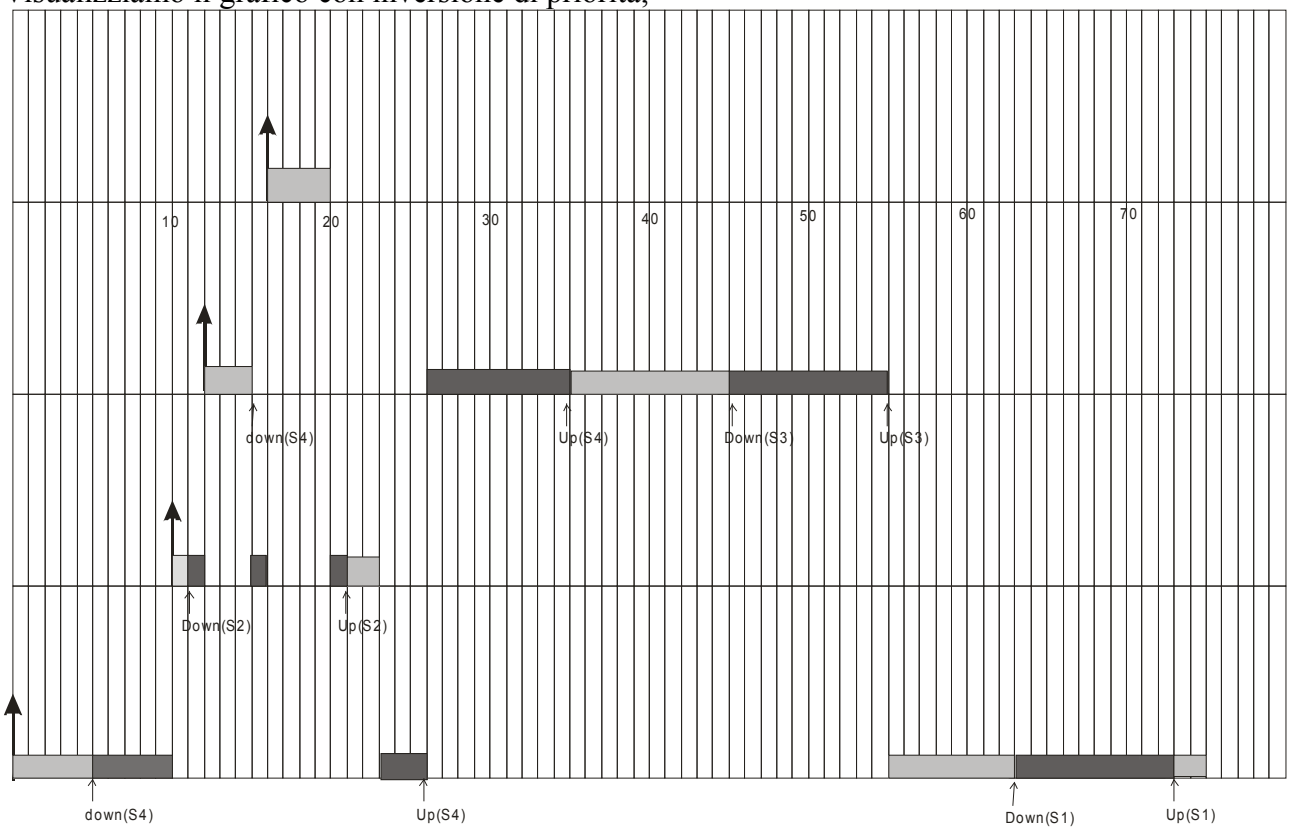
7 **(8pt)** Si supponga che ci siano quattro task che eseguono come segue (4=priorità alta, 1=priorità bassa) utilizzando delle risorse condivise che vengono protette dai semafori S1,S2,S3,S4.
T1 (priorità=4): arrivo all'istante 16; esegue per 4 unità di calcolo; fine
T2 (priorità=3):
arrivo all'istante 12; esegue per 3 unità di calcolo;
wait(S4); esegue in sezione critica complessivamente per 9 unità di calcolo; signal(S4);
esegue per 10 unita' di calcolo;
wait(S3); esegue in sezione critica complessivamente per 10 unita' di calcolo; signal(S3); fine;

T3 (priorità=2):
arrivo all'istante 10; esegue per una unita' di calcolo;
wait(S2); esegue in sezione critica complessivamente per 3 unità di calcolo; signal(S2);
esegue per 2 unita' di calcolo; fine;
T4 (priorità=1):
arrivo all'istante 0; esegue per 5 unita' di calcolo;
down(S4); esegue in sezione critica complessivamente per 8 unita' di calcolo; signal(S4);
esegue per 8 unita' di calcolo;
wait(S1); esegue in sezione critica complessivamente per 10 unita' computazionali; signal(S1);
esegue per 2 unita' computazionali; fine;

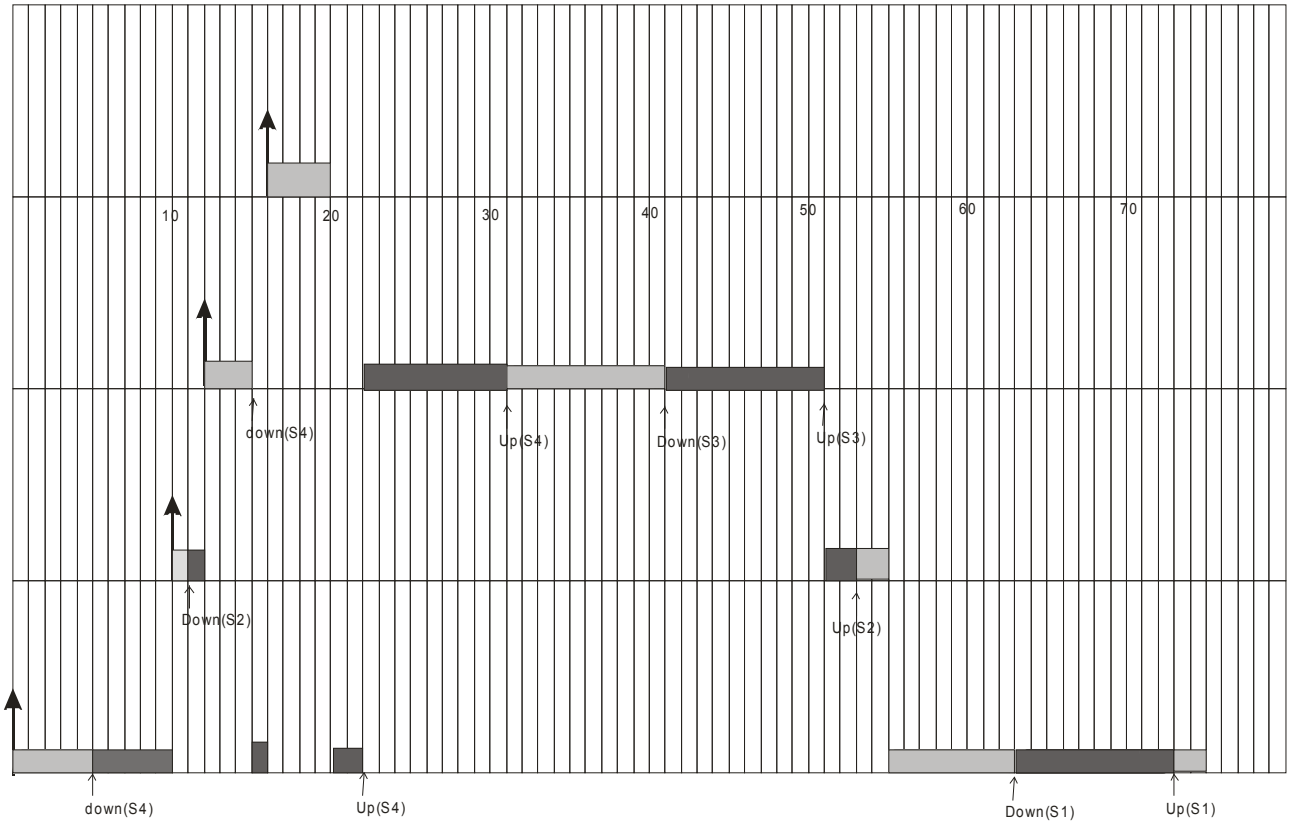
Questi 4 task vengono eseguiti con uno schedulatore Rate Monotonic. Rispondere alle seguenti domande:
7a. Tracciare il grafico di schedulazione senza nessun protocollo per l'accesso a risorse condivise.
7b. Tracciare il grafico usando il protocollo 'Priority inheritance'.
7c. Confrontare il tempo di blocco di T1 nei due casi.

Soluzione

Visualizziamo il grafico con inversione di priorità;



Visualizziamo ora il grafico di schedulazione con l'uso del protocollo di Priority Inheritance (PI):



Dai grafici si vede che:

- T1 non risente di inversioni perchè non usa risorse critiche
- T2 finisce a 55 senza protocollo PI, a 51 con PI
- T3 finisce a 55 con PI, a 23 senza PI
- T4 finisce sempre a 75

8 (1.5pt) Qual'è 'uscita di questo programma POSIX?

```
Test()
{
        int pfd[2];
        char s[100];

        pipe(pfd);

        read(pfd[0], s, 100);
        printf("letto %s\n", s);
        write(pdf[1],' test ',6);
}
```

Soluzione

Il programma si blocca perchè la pipe è vuota: la prima istruzione (lettura dalla pipe) blocca l'esecuzione. (pipe=punto di sincronizzazione)