Politecnico di Milano – Sede di Cremona
Anno Accademico 2017/2018

Architettura dei Calcolatori e Sistemi Operativi

Esame – 29.01.2018

Prof. Carlo Brandolese

Cognome _____ **Nome**

Matricola _____ **Firma**

Istruzioni

1. Scrivere con cura, negli spazi sopra segnati, il proprio cognome, nome, numero di matricola e apporre la firma.
2. È vietato consultare libri, eserciziari, appunti ed utilizzare la calcolatrice e qualunque strumento elettronico (inclusi i cellulari), pena l'invalidazione del compito.
3. Il testo, debitamente compilato, deve essere riconsegnato in ogni caso.
4. Il tempo della prova è di 3 ore

Valutazione

Domanda	Voto	Note
A		
B		
C		
D		
E		
F		

Domanda A

Si implementi in linguaggio assembly la funzione:

```
void strrev( char* srt )
```

che “inverte” la stringa **`str`** sul posto, ovvero senza utilizzare altra memoria se non quella richiesta dalla stringa stessa. Per inversione si intende la riscrittura dei caratteri della stringa in ordine inverso. Si tenga presente che sia la stringa in ingresso, sia quella prodotta dalla funzione sono stringhe C, ovvero terminate dal carattere nullo.

Si traduca quindi in assembly in seguente programma C che utilizza la funzione **`strrev()`**:

```
char* str = "This is the text!";

int main( void )
{
  strrev( str );
  return 0;
}
```

Domanda B

Si consideri il seguente insieme di processi:

Process	Arrival Time (T_A)	Execution Time (T_E)
P1	0	8
P2	2	2
P3	2	5
P4	6	5
P5	10	3

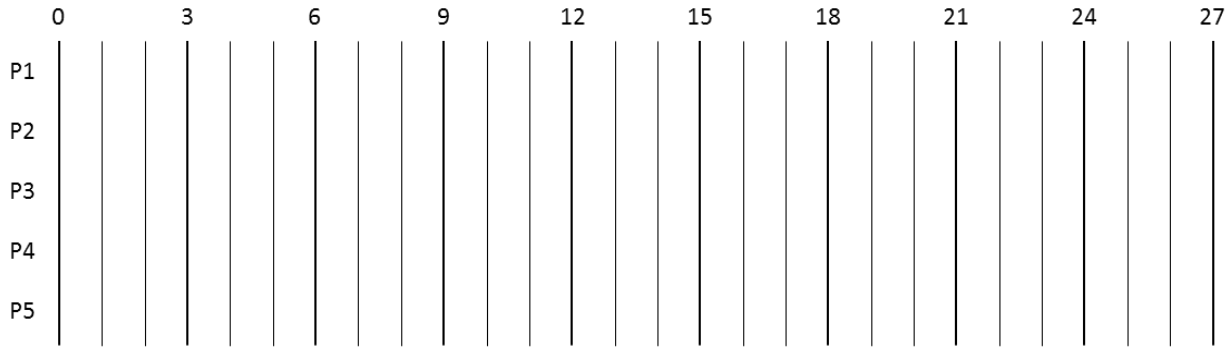
Si esegua lo scheduling di tali processi secondo i due seguenti algoritmi:

- Roud Robin, non-preemptive, con un quanto di tempo pari a 3 unità
- Shortest Remaining Time

Per ognuno dei due casi, quindi, si svolgano i seguenti punti:

- Si indichi il tempo reale di esecuzione di ogni processo
- Si calcoli il tempo di attesa medio T_W dei processi
- Supponendo che un processo P_n abbia una deadline di completamento $T_{D,n}$ data dalla seguente relazione $T_{D,n} = T_{A,n} + 1.5 * T_{E,n}$, si indichino quali processi sono completati entro la deadline in ognuno dei due casi.

Round Robin

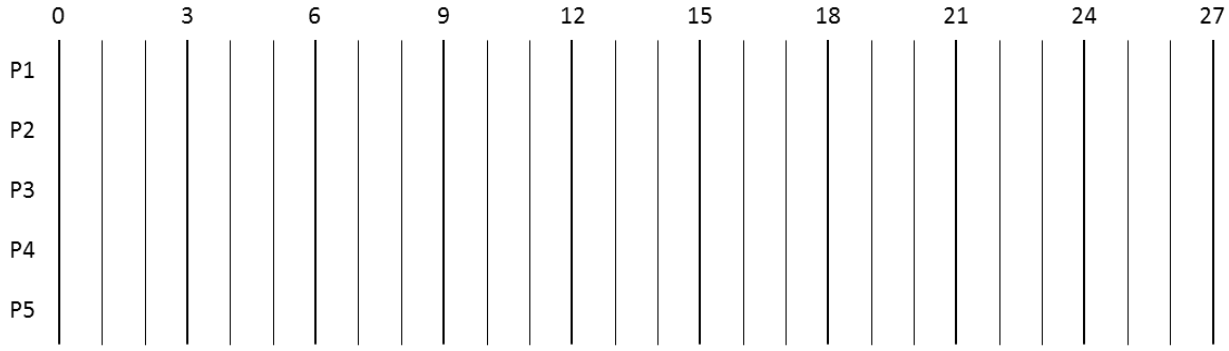


TW =

Processi completati entro la deadline:

Processo	Tempo Reale di esecuzione
P1	
P2	
P3	
P4	
P5	

Shortest Remaining Time



TW =

Processi completati entro la deadline:

Processo	Tempo Reale di esecuzione
P1	
P2	
P3	
P4	
P5	

Domanda C

Si consideri un sistema con uno spazio di indirizzamento di 4 GByte, una cache L0 di primo livello ed una cache L1 di secondo livello. Entrambe le cache sono unificate, ovvero sono utilizzate sia per i dati, sia per le istruzioni. Si ricorda che la cache L0 immediatamente connessa al processore, mentre la cache di secondo livello è connessa tra quella di primo livello e la memoria RAM. Nel caso di L0 miss, la richiesta passa alla cache L1 e solo se anche in questo caso si ha un miss, allora è necessario accedere alla memoria RAM. Le caratteristiche delle due cache sono le seguenti:

	Cache L0	**Cache L1**
Associatività	Mapping diretto	Set associativa a 8 vie
Dimensione totale	32 KB	1 MB
Dimensione linea	128 B	1 KB (per ogni set)
Tempo di accesso	1 ns	4 ns
Hit rate	90%	99%

Sulla base di queste informazioni si indichi la struttura dell'indirizzo visto dalle cache, descrivendo i vari campi e il loro significato.

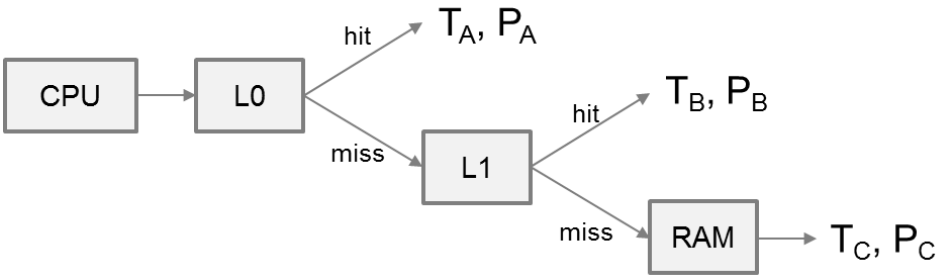
Cache L0

Cache L1

Sapendo che:

- L'accesso alla memoria RAM avviene a parole di 32 bit
- Il tempo di accesso alla RAM in modalità normale è di 50 ns
- Il tempo di accesso alla RAM in modalità burst è
    - 100 ns per la prima parola
    - 10 ns per le parole successive

Si calcoli il tempo di accesso medio alla memoria. A questo scopo si consideri il seguente diagramma che mostra i diversi casi di hit/miss ai diversi livelli di gerarchia di memoria.



Questo diagramma indica i tre tempi T_A, T_B e T_C e le relative probabilità P_A, P_B e P_C. Per procedere al calcolo del tempo medio richiesto, si proceda dapprima al calcolo di tali tempi e delle relative probabilità (nell'ipotesi semplificativa che le probabilità di hit in L0 ed in L1 siano indipendenti).

TA, PA

TB, PB

TC, PC

Tempo di accesso medio alla memoria

Domanda D

Si consideri un microprocessore MIPS dotato di una pipeline standard a cinque stadi **dotata di percorsi di propagazione**, ma senza ottimizzazioni per le istruzioni di salto.

Sia inoltre dato il seguente codice:

```
1.          SGT   R3, R1, R2
2.          ADD   R4, R1. R1
3.          SUB   R5, R4, R2
4.          BEQZ  R3, ELSE
5.          ADD   R6, R4, R0
6.          J     END
7.    ELSE: ADD   R6, R5, R0
8.    END:
```

In primo luogo si simuli l'esecuzione del codice inserendo stalli ove necessario, e si calcoli il numero di cicli di clock necessari per eseguire il codice, e il CPI. Per svolgere l'esercizio mostrare mediante uno schema come le istruzioni attraversano i vari stadi del MIPS in corrispondenza dei cicli di clock.

N° Istr.	**Cicli di Clock**														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

CPI

Si supponga quindi di inserire le ottimizzazioni di **valutazione anticipata del salto** e di **speculazione branch not taken** all'interno dell'architettura. Si simuli nuovamente il codice inserendo stalli ove necessario, e si calcoli il numero di cicli di clock necessari per eseguire il codice, e il CPI. Per svolgere l'esercizio mostrare mediante uno schema come le istruzioni attraversano i vari stadi del MIPS in corrispondenza dei cicli di clock.

N° Istr.	**Cicli di Clock**														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

CPI

Domanda E

Dato il seguente programma C:

```
#include ...

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
sem_t semaphore;

int shared = 5;

void* thread1( void* arg )
{
      int var1 = 3;

      pthread_mutex_lock( &mutex );
      shared = shared - var1;          BREAKPOINT A
      pthread_mutex_unlock( &mutex );

      sem_wait( &semaphore );

      var1 = var1 + shared;            BREAKPOINT B
}

void* thread2( void* arg )
{
      int var2 = 2;

      pthread_mutex_lock( &mutex );
      shared = shared + var2;          BREAKPOINT C
      pthread_mutex_unlock( &mutex );

      sem_post( &semaphore );

}

int main()
{
      pthread_t th1, th2;

      sem_init      ( &semaphore, 0, 0 );

      pthread_create( &th1, NULL, &thread1, NULL );
      pthread_create( &th2, NULL, &thread2, NULL );

      pthread_join  ( th1, NULL );
      pthread_join  ( th2, NULL );


      sem_destroy   ( &semaphore );

      return 0;
}
```

Si svolgano i seguenti punti:

1. Si completi la seguente tabella, riportando lo stato delle variabili **immediatamente dopo** il breakpoint indicato. Nel caso si passi più di una volta da un breakpoint considerare ogni sua esecuzione nel rispondere. Si utilizzi la seguente classificazione:
   - "ESISTE" se la variabile certamente esiste
   - "NON ESISTE" se sicuramente la variabile non esiste
   - "PUO' ESISTERE" se potrebbe esistere oppure non esistere

Breakpoint	var1	var2	shared
Breakpoint A			
Breakpoint B			
Breakpoint C			

2. Si completi la tabella seguente indicando tutti i possibili valori che le variabili riportate possono assumere subito dopo i breakpoint specificati.

Breakpoint	var1	var2	shared
Breakpoint A			
Breakpoint B			
Breakpoint C			

3. Si immagini di rimuovere il mutex sugli accessi alla variabile *shared*. Quale comportamento si potrebbe ottenere in questo caso? Si fornisca una breve descrizione

Domanda F

Si descriva nel modo più preciso e chiaro possibile il concetto di semaforo e si spieghi per quale ragione non può essere utilizzato per la sincronizzazione tra processi.