Politecnico di Milano – Sede di Cremona
Anno Accademico 2020/2021

Architettura dei Calcolatori e Sistemi Operativi

Esame – 07.07.2021

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 vuole implementare un modulo per la gestione di un semplice stack di dimensione fissa i cui elementi sono di tipo intero. L'implementazione, tuttavia, deve essere thread-safe, cioè deve **funzionare correttamente anche quando utilizzata da più thread concorrenti**. Le funzioni che il modulo deve implementare sono le seguenti:

`int pop(void)`	Restituisce il valore alla cima dello stack e lo rimuove
`void push(int n)`	Impila il nuovo valore n sullo stack
`int top(void)`	Restituisce il valore alla cima dello stack, senza rimuoverlo

A tale scopo si considerino le seguenti assunzioni:

1. Lo stack è inizialmente vuoto
2. Gli elementi dello stack sono memorizzati in un array privato del modulo
3. La dimensione massima dello stack è definita dalla macro **`STACK_SIZE`**
4. Se lo stack è pieno la funzione **`push()`** non impila il nuovo elemento
5. Se lo stack è vuoto la funzione **`pop()`** restituisce il valore 0

```
#define STACK_SIZE  1000

static int stack[STACK_SIZE];
static int sp = 0;
static pthread_mutex_t mutex;

void push( int n ) {
     pthread_mutex_lock(&mutex);
     if( sp < STACK_SIZE ) {
          stack[sp++] = n;
     }
     pthread_mutex_unlock(&mutex);
}

int pop( void ) {
     int value = 0;
     pthread_mutex_lock(&mutex);
     if( sp > 0 ) {
          value = stack[--sp];
     }
     pthread_mutex_unlock(&mutex);
     return value;
}

int top( void )
{
     int value = 0;
     pthread_mutex_lock(&mutex);
     if( sp > 0 ) {
          value = stack[sp];
     }
     pthread_mutex_unlock(&mutex);
     return value;
}
```

Domanda B

Si scrivano in codice assembly MIPS le funzioni:

```
void push( int n );

int pop( void );
```

che eseguono le tipiche operazioni di gestione dello stack. Per l'implementazione si parta dalla seguente definizione dei sue simboli **SP** (stack pointer) e **STACK** memoria riservata allo stack. Per semplicità, si ignorino le condizioni di errore relative alla situazione stack vuoto e di stack pieno.

```
            .data

SP:         .word       0
STACK:      .space      4096


            .text



push:   la   $t0, SP          # &SP
        la   $t1, STACK       # &STACK
        lw   $t2, 0($t0)      # SP
        add  $t1, $t1, $t2    # &STACK[SP]
        sw   $a0, 0($t1)      # STACK[SP] = arg
        addi $t2, $t2, 4      # SP++
        sw   $t2, 0($t0)      # save SP
        jr   $ra              # return

pop:    la    $t0, SP         # &SP
        la    $t1, STACK      # &STACK
        lw    $t2, 0($t0)     # SP
        addi  $t2, $t2, -4    # SP++
        add   $t1, $t1, $t2   # &STACK[SP]
        lw    $v0, 0($t1)     # retval = STACK[SP]
        sw    $t2, 0($t0)     # save SP
        jr    $ra             # return
```

Domanda C

Si consideri una gerarchia di memoria composta da:

- Memoria centrale: 4GB, indirizzabile a parole di 16 bit
- Cache istruzioni: 64K, diretta, linee da 128B
- Cache dati: 8K, completamente associativa, linee da 128B

Tali memorie presentano i seguenti tempi di accesso:

- Accesso alle cache: 1 ciclo di clock
- Accesso alla memoria centrale in modalità normale: 100 cicli di clok
- Accesso alla memoria centrale in modalità burst: 200 cicli di clock per la prima parola, 50 cicli di clock per le parole a indirizzi successive (modalità burst).

Si consideri inoltre la situazione in cui:

- Hit rate della memoria istruzioni pari al 98%
- Hit rate della memoria dati pari al 99.5%

Sulla base di queste informazioni:

1. Indicare la struttura degli indirizzi di memoria per le memorie cache

4GB -> 32 bit

ICACHE
#index = log2(64KB / 128)= log2(2^16 / 2^7) = 9
#offset = log2(128) = 7
#tag = 32 – 9 – 7 = 16

DCACHE
#offset = log2(128) = 7
#tag = 32 – 7 = 25

2. Calcolare il tempo medio di accesso alle due cache

Le linee della ICACHE e della DCACHE sono uguali pertanto:

Tmiss = 1cc + 200cc + (128 / 2 – 1) * 50cc = 3351cc

Per la ICACHE:
Tave = 1cc * 0.98 + 3351cc * 0.02 = 68cc

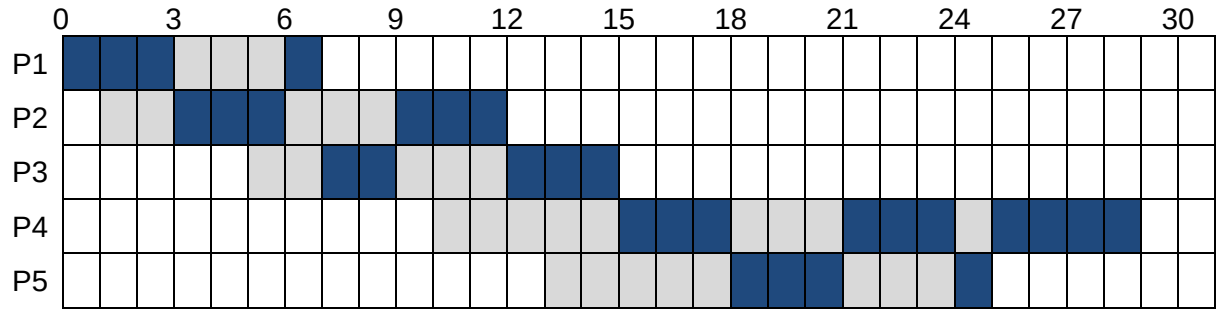
Per la DCACHE
Tave = 1cc * 0.995 + 3351cc * 0.005 = 17.75 cc

Domanda D

Si considerino i seguenti processi:

Processo	Arrival Time	Execution time
P1	0	4
P2	1	6
P3	5	5
P4	10	10
P5	13	4

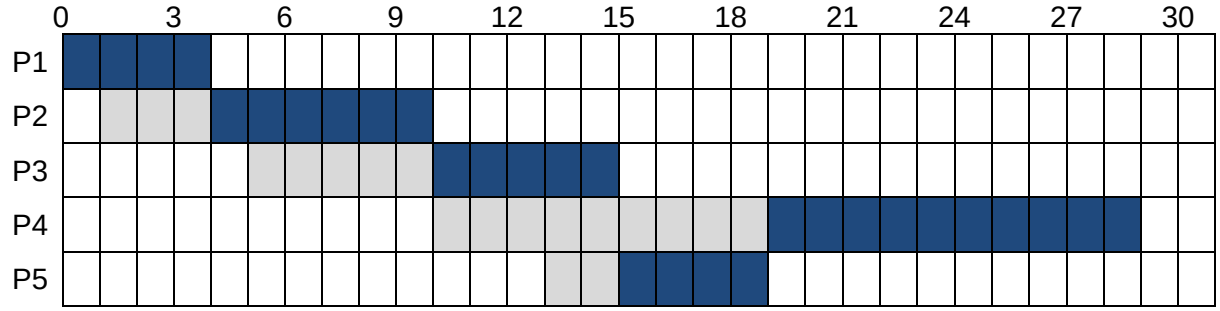
Si realizzi lo scheduling secondo l'algoritmo Round Robin con quanto di tempo pari a 3 tick di sistema, sapendo che al momento del context switch vengono dapprima accodati i processi uscenti sospesi, quindi i processi nuovi. Si calcolino il tempo di attesa massimo e medio.



$T_{MAX} =$ 9

$T_{AVE} =$ (3 + 5 + 5 + 9 + 8) / 5 = 6

Si realizzi lo scheduling secondo l'algoritmo Shortest Remaining Time e si calcolino anche in questo caso il tempo di attesa massimo e medio.



$T_{MAX} =$ 9

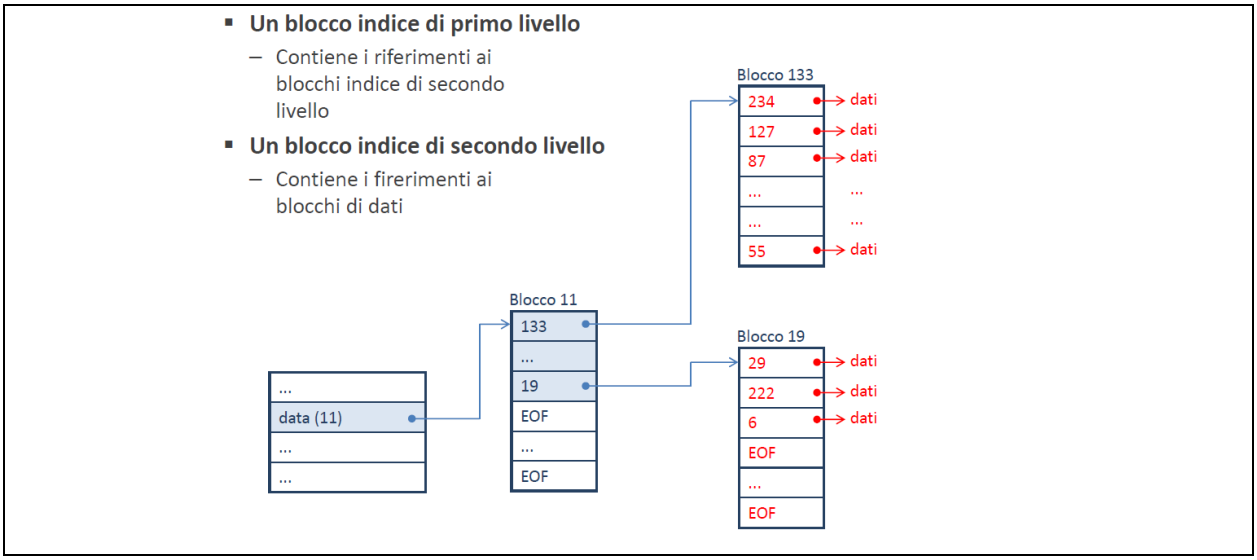
$T_{AVE} =$ (0 + 3 + 5 + 9 + 2) / 5 = 3.8

Domanda E

Si un filesystem organizzato come descritto nel seguito.

- Dimensione blocco: 512 byte
- Dimensione indirizzi: 2 byte
- Allocazione: Indicizzata a due livelli

Si rappresenti graficamente la struttura di un file.



Si calcoli la dimensione massima del volume

Con indirizzi di 2 byte si possono indirizzare 2^16 blocchi.
Ogni blocco ha dimensione 512 = 2^9 byte
Il volume ha dimensione massima 2^16 *2^9 = 2^25 = 32 MB

Si calcoli la dimensione massima di un file

Ogni blocco contiene 512 / 2 = 2^9 / 2 = 2^8 indici
Si hanno pertanto 2^8 blocchi di secondo livello e (2^8)*(2^8) blocchi dati
La dimensione massima di un file è perciò (2^8)*(2^8)*(2^9) = 2^25 = 32 GB

NOTA
Siccome il volume è della stessa dimensione, in realtà anche un unico file non potrebbe avere tale dimensione bensì, al più, 32GB meno la dimensione dei 2^8+1 blocchi indice.

Si determini il numero di blocchi cui è necessario accedere per leggere 2KB di dati di un file a partire dalla posizione 1100 dall'inizio del file stesso.

Per accedere a 2KB sono necessari 5 blocchi poiché l'inizio della zona da leggere non è allineata ad un multiplo di 512 byte.
Si deve inoltre accedere al blocco indice di primo livello ed al primo blocco indice di secondo livello.

Il numero totale di accessi è pertanto 2 + 5 = 7.

Domanda F

Si descriva il concetto di “register forwarding” e la motivazione per cui è utile inserirlo nella pipeline MIPS. Si riporti un diagramma, anche semplificato, di come tale tecnica modifica la pipeline.