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

Architettura dei Calcolatori e Sistemi Operativi

Esame – 17.06.2021

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 realizzi un programma che mediante 8 thread paralleli conta il numero di linee in un testo molto lungo e stampa tale numero sullo standard output. Una linea è identificata dalla presenza del carattere di newline **`\n`**. Si supponga che il testo sia memorizzato nella variabile **`text`** e la sua lunghezza nella variabile **`length`**, entrambe locali alla funzione **`main()`**.
Nel realizzare tale programma si ponga attenzione a massimizzare la concorrenza.

```
int  pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine) (void *), void *arg);
int  pthread_join(pthread_t thread, void **retval);
void pthread_exit(void *retval);
int  sem_post(sem_t *sem);
int  sem_wait(sem_t *sem);
int  sem_init(sem_t *sem, int pshared, unsigned int value);
int  pthread_mutex_lock(pthread_mutex_t *mutex);
int  pthread_mutex_unlock(pthread_mutex_t *mutex);
int  pthread_mutex_init(pthread_mutex_t *restrict mutex,
                        const pthread_mutexattr_t *restrict attr);
```

```
char* text = ...;
int   length = strlen(text);

int total_lines;
pthred_mutex_t mutex;

void* count_lines( void* arg )
{
  char* ptr   = (char*)arg;
  int   lines = 0;

  for( int i = 0; i < length/8; i++ ) {
    if( ptr[i] == '\0' ) breaK
      if( ptr[i] == '\n' ) lines++;
  }

  pthread_mutex_lock( &mutex );
  total_lines += lines;
  pthread_mutex_lock( &mutex );
}
```

```
int main( int argc, char** argv )
{
  pthread_t tid[8];

  for( int i = 0; i < 8; i++ ) {
    pthread_create( &tid[i], NULL, count_lines, (void*)(text+i*length/8) );
  }

  for( int i = 0; i < 8; i++ ) {
    pthread_join( tid[i], NULL );
  }

  printf( "Lines: \n", total_lines );

}
```

Domanda B

Si scriva in codice assembly MIPS la funzione:

`int prod( int* x, int* y, int n )`

che prende in ingresso due vettori **`x`** ed **`y`** di **`n`** elementi interi e restituisce il valore:

$$s = \sum_{i=0}^{n-1} x_i \cdot y_i$$

La funzione può essere realizzata come segue:

```
prod: add   $t0, $a0, $zero         # pointer to x
      add   $t1, $a1, $zero         # pointer to y
      add   $t2, $zero, $zero       # index
      add   $t3, $zero, $zero       # total

loop: beq   $t2, $a2, end
      ld    $t4, 0($t0)             # x[i]
      ld    $t5, 0($t1)             # y[i]

      mult  $t4, $t5
      mflo  $t4                     # t4=x[i]*y[y]

      add   $t3, $t3, $t4           # accumulates result

      addi  $t2, $t2, 1             # increment
      addi  $t0, $t0, 4             # move pointer
      addi  $t1, $t1, 4             # move pointer

      b     loop

end:  move  $v0, $t3
      jr    $ra
```

le dichiarazioni ed un esempio di utilizzo sono riportati di seguito:

```
      .data

x:    .word 1 2 3 4
y:    .word 2 4 6 8

      .text

main: la    $a0, x
      la    $a1, y
      add   $a2, $zero, 4
      jal   prod                    # call

      move  $a0, $v0                # print
      li    $v0, 1
      syscall
```

Domanda C

Si consideri una gerarchia di memoria composta da:

- Memoria centrale: 4GB, indirizzabile a parole da 32 bit
- Cache istruzioni: 1MB, diretta, linee da 512B
- Cache dati: 1MB, set associativa a 4 vie, linee da 64B per ogni set

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: 125 cicli di clock per la prima parola, 25 cicli di clock per le parole a indirizzi successive (modalità burst).

Si consideri inoltre la situazione in cui:

- Il sistema esegue un programma in cui in media il 20% delle istruzioni richiede un accesso alla memoria (sia esso il lettura o scrittura).
- Il miss rate della cache dati è del 2%
- Il miss rate della cache istruzioni è dello 0.5%

Sulla base di queste informazioni:

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

4GB -> 32 bit

ICACHE
#index = log2(1MB / 512) = 11
#offset = log2(512) = 9
#tag = 32 – 11 – 9 = 12

DCACHE
#index = log2(1MB / (64 *4)) = 12
#offset = log2(64) = 6
#tag = 32 – 12 – 6 = 14

2. Calcolare il tempo necessario per caricare un blocco in caso di fallimento (miss time)

ICACHE
Tmiss,I = 1cc + 125cc + (512 / 4 – 1) * 25cc = 3301cc

DCACHE
Tmiss,D = 1cc + 125cc + (64 / 4 – 1) * 25cc = 501 cc

3. Calcolare il tempo medio di accesso alla memoria

Tave,I = 1cc * 0,995 + 3301 * 0,005 = 17,5 cc
Tave,D = 1cc * 0,98 + 501 * 0,02 = 11 cc

Tave = Tave,I + 0,2 Tave,D = 17,5cc + 0,2 * 11cc = 19,7

Domanda D

Si consideri un file system Linux organizzato come mostra la figura seguente.

I-node list:
```
  < 0,dir,  3>  < 3,dir,27>  < 4,dir,  19>  < 7,dir,50>
  < 9,norm,24>  <13,dir,33>  <15,norm, 70>  <17,dir,47>
  <20,norm,{120,121,122,123,124,125,126,127,128}>
  <23,norm,78>
```

Blocco003: … < 4,temp> < 9,pippo> …
Blocco013: … < 7,mydir> <15,prova> …
Blocco019: … <13,data> <17,documents> …
Blocco033: … <20,file1> <23,file2> …

Blocco120: … dati …
...
Blocco128: … dati …

Si consideri inoltre il programma seguente:

```
int main() {
  int  fd;
  char buf[1024];

  fd = open( "/tmp/data/file1.txt", O_RDONLY );

  if( fork() == 0 ) {
    read( fd, buf, 600 );
    close( fd );
    exit( 0 );
  }

  wait( NULL );
  close( fd );
  exit( 0 );
}
```

Per ciascuna delle seguenti chiamate di sistema, si completi la tabella riportando la sequenza di accesso agli i-node ed ai blocchi in memoria principale (M) o su disco (D).

Chiamata di sistema	Sequenza di accessi
`fd = open(...)`	INODE 0 -> BLOCCO 3 (D) INODE 4 -> BLOCCO 19 (D) INODE 13 -> BLOCCO 33 (D) INODE 20 -> BLOCCO 120 (D)
`read(...)`	INODE 20 -> BLOCCO 120 (D), BLOCCO 1201 (D)

Chiamata di sistema	Sequenza di accessi
`close(...)` nel figlio	Nessun accesso
`close(...)` nel padre	Nessun accesso

Domanda E

Si consideri un sistema di memoria con uno spazio di indirizzamento virtuale di 1MB ed una dimensione di pagina pari a 2KB.

1. Si indichino le dimensioni in bit di:

   Indirizzo virtuale: log2(1MB) = 20

   Numero di pagina virtuale: log2(1MB / 2KB) = 9

   Offset: log2(2K) = 11

2. Si completi la seguente tabella, riportando numero di pagina virtuale e piazzamento sia in forma binaria, sia esadecimale.

Indirizzo virtuale	Numero di pagina virtuale		Offset	
	Hex	Bin	Hex	Bin
0x80CBA	0x101	1 0000 0001	0x4BA	100 1011 1010
0x49FF2	0x093	0 1001 0011	0x7F2	111 1111 0010
0x0003A	0x000	0 0000 0000	0x03A	000 0011 1010
0xF0800	0x1E1	1 1110 0001	0x000	000 0000 0000
0xCCCCC	0x199	1 1001 1001	0x4CC	100 1100 1100

Domanda F

Si spieghino i concetti di TLB miss e di page fault, indicando i vari elementi software e hardware coinvolti. Si descrivano inoltre, per ognuno dei due casi, le conseguenze in termini di tempi di accesso ai dati.

Si vedano il testo e le slide.