Matricola: _____

Nome: _____

Cognome: _____

ESAME (SIMULAZIONE)
Programmazione Avanzata e Parallela

9 gennaio 2024

L'esame consiste di 10 domande a risposta multipla sugli argomenti del corso. Ogni domanda può ricevere un punteggio massimo di *due* punti. Affinché una risposta sia considerata valida la scelta *deve essere motivata*. Una risposta errata o non motivata riceverà *zero* punti.

Domanda 1

Si supponga di avere il seguente Makefile:

```
all: a.o b.o c.o
    gcc a.o b.o c.o -o program

a.o: a.c
    gcc -c a.c

b.o: b.c g.h
    gcc -c b.c

c.o: c.c g.h
    gcc -c c.c
```

Si supponga inoltre di aver eseguito in precedenza il comando `make` e, successivamente, di aver modificato il file `g.h`. Eseguendo nuovamente il comando `make` quali saranno i comandi eseguiti?

☐
```
gcc -c b.c
gcc -c c.c
```

☐
```
gcc -c a.c
gcc -c b.c
gcc -c c.c
gcc a.o b.o c.o -o program
```

☐
```
gcc -c b.c
gcc -c c.c
gcc a.o b.o c.o -o program
```

☐ Nessun comando viene eseguito

Domanda 2

Si supponga di avere del codice C nella seguente forma:

```
for (int i = 0; i < n; i++) {
    if (v[i] > 0) {
        x += v[i];
    }
    if (v[i] % 2 == 0) {
        y += v[i];
    }
}
```

e si assuma di avere variabili `x, y, v` definite e del tipo corretto con i valori di `v` uniformemente distribuiti tra 0 e 100.000 e in ordine casuale nell'array.
Quale dei due `if` *potrebbe* essere utile convertire in una versione branchless?
Si assuma che il compilatore non sia in grado di generare in automatico codice branchless.

☐ Il primo ☐ Il secondo

☐ Entrambi ☐ Nessuno

Domanda 3

Siano dati i due frammenti di codice C:

```
for (int i = 0; i < n; i++) {
  sum += v[i];
}
```

e

```
while (h != NULL) {
  sum += h->val;
  h = h->next;
}
```

dove `v` è un vettore di n elementi e `h` un puntatore a un nodo di una lista concatenata di n elementi (ogni nodo contiene un valore `val` e un puntatore a nodo successivo `next`).
Quale dei due frammenti ci aspettiamo sia più veloce nell'eseguire su un processore moderno?

☐ Circa uguali nel tempo di esecuzione

☐ Il primo

☐ Il secondo

☐ Il secondo ma solamente se il compilatore può usare le istruzioni SIMD

Domanda 4

Data le seguenti strutture:

```
struct S1 {
  int8_t a;
  int8_t b;
  int32_t c;
};

struct S2 {
  int8_t a;
  int32_t c;
  int8_t b;
};
```

in cui `intn_t` indica un intero con segno di n bit, quali delle seguenti affermazioni è vera?

☐ È sempre vero che `sizeof(struct S1) == sizeof(struct S2)`

☐ La dimensione di `struct S1` è di 6 byte

☐ Le dimensioni delle due strutture potrebbero differire

☐ Possiamo copiare nei campi corretti il contenuto di una `struct S1` in una `struct S2` con `memcpy`

Domanda 5

Quale delle seguenti affermazioni riguardo l'I/O in C è sbagliata?

- ☐ Il buffer in cui sono salvati i dati letti con `fread` deve avere almeno dimensione pari al numero di elementi letti moltiplicato per la loro dimensione
- ☐ `getc` può ritornare `EOF`
- ☐ `fseek` permette di spostare la posizione di lettura/scrittura sia in modo assoluto (rispetto a inizio o fine del file) che relativo (rispetto alla posizione corrente)
- ☐ Effettuare `mmap` richiede di avere abbastanza memoria fisica a disposizione per mantenere l'intero contenuto del file

---

Domanda 6

Dato il seguente codice C facente uso di OpenMP:

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int * random_binary_vector(int n) { /* ... */ }

int main(int argc, char * argv[])
{
  const int n = 100000;
  int * v = random_binary_vector(n);
  int count[2] = {0, 0};
#pragma omp parallel for
  for (int i = 0; i < n; i++) {
    count[v[i]]++;
  }
  printf("%d zeros, %d ones\n", count[0], count[1]);
  return 0;
}
```

Si supponga che la funzione `random_binary_vector` sia correttamente definita e ritorni un vettore di valori che sono o 0 o 1.
Quale è il problema del precedente codice?

- ☐ Il ciclo `for` viene eseguito n volte per ogni thread invece di n volte in totale
- ☐ Essendoci un solo array `count` non vi sarà alcuno speedup utilizzando più thread
- ☐ L'accesso a `count` porta a delle race conditions e il risultato non sarà corretto
- ☐ Il codice è corretto

Domanda 7

Dato il seguente codice Python:

```
class A:

    def m(self):
        print("A")


class B(A):
    pass


class C(B):

    def m(self):
        print("C")


class D(B):
    pass


class E(D):

    def m(self):
        print("E")


x = D()
x.m()
```

quale è il valore stampato a schermo?

☐ `A`

☐ `C`

☐ `E`

☐ Viene generata una eccezione perché `D` non definisce `m`

Domanda 8

Dato il seguente codice Python:

```
def f(g, h, x):
    def c(y):
        return g(h(x), h(y))
    return c


func = f(lambda x, y: x + y, lambda x: 3*x, 2)
```

Quale è il valore ritornato da `func(4)`:

☐ Non è possibile chiamare una variabile

☐ `18`

☐ `12`

☐ Un output nella forma di `<function f.<locals>.c at 0x102cb7c70>`

Domanda 9

Dato il seguente codice Python:

```
class MyException(Exception):
    pass


def f():
    try:
        return g()
    except Exception:
        print("Exception catched by f")
    except MyException:
        print("Exception catched by f")


def g():
    try:
        h()
    except MyException as e:
        print("MyException catched by g")
        raise e
    except Exception:
        print("Exception catched by g")


def h():
    raise MyException()
```

Cosa viene stampato a schermo chiamando `f()`?

☐ `MyException catched by g`

☐ `Exception catched by g`

☐ `MyException catched by g`
`Exception catched by f`

☐ `MyException catched by f`
`MyException catched by g`

Domanda 10

Dato il seguente codice Python:

```
def f(x):
    while True:
        yield x
        x += 1


h = f(4)
for i in range(5):
    print(next(h))
```

Quali sono i valori stampati a schermo?

☐ `4 4 4 4 4`

☐ La chiamata `f(4)` non termina mai a causa di un loop infinito

☐ `4 5 6 7 8 9 10 ...` non terminando mai

☐ `4 5 6 7 8`