POLITECNICO
DI TORINO

Programmazione di sistema

Esame 20 Giugno 2022 - Programming

Iniziato	lunedì, 20 giugno 2022,
Terminato	lunedì, 20 giugno 2022,
Tempo impiegato	
Valutazione	

Domanda 1

Completo

Punteggio ottenuto ,00 su 3,00

Si definisca il concetto di Smart Pointer, quindi si fornisca un esempio (Rust o C++) che ne evidenzi il ciclo di vita.

'''

'''

Commento:

Domanda 2

Completo

Punteggio ottenuto ,00 su 3,00

Si illustrino le differenze nel linguaggio Rust tra **std::channel()** e **std::sync_channel()**, indicando quali tipi di sincronizzazione i due meccanismi permettono.

Commento:

Domanda 3

Completo

Punteggio ottenuto ,00 su 3,00

Dato il seguente frammento di codice Rust (ogni linea è preceduta dal suo indice)

```
1.  struct Point {
2.      x: i16,
3.      y: i16,
4.  }
5.
6.  enum PathCommand {
7.     Move(Point),
8.     Line(Point),
9.     Close,
10. }

11. let mut v = Vec::<PathCommand>::new();
12. v.push(PathCommand::Move(Point{x:1,y:1}));
13. v.push(PathCommand::Line(Point{x:10, y:20}));
14. v.push(PathCommand::Close);
15. let slice = &v[..];
```

Si descriva il contenuto dello stack e dello heap al termine dell'esecuzione della riga 15.

Commento:

Domanda 4

Completo

Punteggio ottenuto ,00 su 6,00

Un paradigma frequentemente usato nei sistemi reattivi è costituito dall'astrazione detta **Looper**.

Quando viene creato, un **Looper** crea una coda di oggetti generici di tipo **Message** ed un thread. Il thread attende - senza consumare cicli di CPU - che siano presenti messaggi nella coda, li estrae a uno a uno nell'ordine di arrivo, e li elabora. Il costruttore di **Looper** riceve due parametri, entrambi di tipo (puntatore a) funzione: **process**(…) e **cleanup**(). La prima è una funzione responsabile di elaborare i singoli messaggi ricevuti attraverso la coda; tale funzione accetta un unico parametro in ingresso di tipo **Message** e non ritorna nulla; La seconda è funzione priva di argomenti e valore di ritorno e verrà invocata dal thread incapsulato nel **Looper** quando esso starà per terminare.

Looper offre un unico metodo pubblico, thread safe, oltre a quelli di servizio, necessari per gestirne il ciclo di vita: **send**(msg), che accetta come parametro un oggetto generico di tipo **Message** che verrà inserito nella coda e successivamente estratto dal thread ed inoltrato alla funzione di elaborazione. Quando un oggetto **Looper** viene distrutto, occorre fare in modo che il thread contenuto al suo interno invochi la seconda funzione passata nel costruttore e poi termini.

Si implementi, utilizzando il linguaggio Rust o C++, tale astrazione tenendo conto che i suoi metodi dovranno essere *thread-safe*.

```
    }


  }
}
```