5 luglio 2021

Si implementi in linguaggio Rust una struttura generica Buffer<T> che modelli una struttura dati condivisa tra due thread concorrenti, uno produttore e uno consumatore. La struttura deve consentire al produttore di inserire valori e notificare la terminazione della produzione, mentre il consumatore può richiedere valori in modalità FIFO, rispettando le seguenti regole:

- **Metodo next(...)**: Il thread produttore utilizza questo metodo per aggiungere un nuovo valore al buffer. Se è stata invocata una chiamata a terminate() o fail(...), il metodo deve fallire lanciando un'eccezione e il buffer deve rimanere inalterato.
- **Metodo terminate()**: Il thread produttore utilizza questo metodo per notificare che non saranno disponibili ulteriori valori. Dopo questa chiamata, il buffer non accetta più nuovi valori. Se non ci sono valori nel buffer, il metodo consume() deve restituire None.
- **Metodo fail(...)**: Il thread produttore utilizza questo metodo per notificare un errore e indicare che non saranno disponibili ulteriori valori. Dopo questa chiamata, il buffer non accetta più nuovi valori. Se non ci sono valori nel buffer, il metodo consume() deve restituire un errore.
- **Metodo consume()**: Il thread consumatore utilizza questo metodo per prelevare un valore dal buffer in modalità FIFO. Se non ci sono valori disponibili, il metodo si blocca in attesa di nuovi valori o di una condizione di terminazione, senza consumare cicli di CPU. Se è stato invocato terminate() e non ci sono valori, restituisce None. Se è stato invocato fail(...) e non ci sono valori, rilancia l'eccezione specificata.

```
impl <T: Send> Buffer<T> {
      pub fn new() -> Self;

    // Metodi relativi alla produzione di valori
      pub fn next(&self, value: T);
      pub fn terminate(&self);
      pub fn fail(&self, error: Box<dyn Any + Send>);

    // Metodo relativo al consumo di valori
      pub fn consume(&self) -> Result<Option<T>, Box<dyn Any + Send>>;
}
```