PDS - API Malnati ottobre 2021

Come viene implementato dal compilatore C++ il comportamento polimorfico degli oggetti? Si descriva il meccanismo messo in atto nel caso di gerarchie di ereditarietà, mettendo in evidenza costi e vantaggi introdotti.

E' possibile in C++, mediante l'uso della programmazione generica, ottenere una forma di polimorfismo senza ricorrere all'uso di metodi virtuali? Si argomenti la risposta con un esempio.

Un sistema concorrente può essere implementato creando più thread nello stesso processo, creando più processi basati su un singolo thread o basati su più thread.

Si mettano a confronto i corrispettivi modelli di esecuzione e si evidenzino pregi e difetti in termini di robustezza, prestazioni, scalabilità e semplicità di sviluppo di ciascuno di essi.

La classe generica ***Processor<T>*** consente ad un insieme di thread produttori di inviare oggetti istanza del tipo T (che si assume copiabile) ad un thread consumatore, incapsulato in ciascuna istanza della classe stessa, ed avviato all'atto della sua creazione. Tale thread consumatore ha il compito di elaborare ciascun oggetto ricevuto servendosi di una funzione fornita come parametro del costruttore.

Il ***costruttore*** di tale classe riceve, come parametro, un oggetto di tipo ***std::function<void(T)>*** e provvederà ad avviare un thread, tenuto rigorosamente all'interno all'istanza della classe, per elaborare, in modo asincrono nell'ordine in cui sono stati ricevuti, gli oggetti di tipo T via via inviati dai produttori.

Tale classe offre ai produttori il metodo ***void send(T t)***, mediante il quale possono sottomettere un oggetto da elaborare (tale oggetto potrà essere posto temporaneamente in una coda, in attesa che il thread consumatore lo elabori) ed il metodo ***void close()***, mediante il quale determinano la fine dell'accettazione di nuovi elementi. L'invocazione di tale metodo ha due conseguenze:

- non sarà più possibile inviare dati (eventuali chiamate a send(...) lanceranno un'eccezione);
- il metodo non ritornerà fino a che il thread secondario non avrà finito di elaborare tutti i dati già ricevuti ed sarà correttamente terminato.

Si implementi la classe generica di seguito proposta utilizzando le funzionalità offerte dal linguaggio C++11, aggiungendo le parti eventualmente necessarie per ottenere il comportamento descritto ed avendo cura di rispettare tutti i vincoli legati all'elaborazione concorrente.

```
template<typename T>
class Processor {
public:
    Processor(std::function<void(T)> f);
    ~Processor();
   void send(T t);
   void close();
};
```
