# Programmazione di sistema

### Anno accademico 2017-2018

## Esercitazione 5

Un ciclo dei messaggi, o message-loop, è un tipico costrutto di programmazione adottato in presenza di interfacce grafiche o di sistemi reattivi che intendono prediligere l'esecuzione sincrona di call-back nell'ambito di un singolo thread pur offrendo la possibilità di reagire allo scorrere del tempo.
Il costrutto può essere implementato sotto forma di classe che incapsula un thread, una coda di messaggi basati su priorità e le necessarie primitive di sincronizzazione volte a garantire la correttezza dell'esecuzione.
La classe offre i seguenti metodi thread-safe:

- **`void startup();`**
  Tale metodo ha effetto solo la prima volta che viene invocato e determina l'avvio del thread responsabile del ciclo di elaborazione dei messaggi
- **`void shutdown();`**
  Tale metodo comporta lo svuotamento della coda e la distruzione del thread, se è mai stato avviato tramite il metodo precedente. Il metodo ha effetto una volta sola e viene invocato anche dal distruttore della classe. Dopo la sua invocazione, eventuali tentativi di inserire nuovi messaggi nella coda con i metodi successivi, lanciano un'eccezione.
- **`void post_task(std::shared_ptr< std::packaged_task<void()> > pt);`**
  Inserisce nella coda dei messaggi un task da svolgere. Quando il thread incapsulato lo estrarrà, provvederà ad invocarlo. Se questo metodo viene invocato dopo shutdown(), lancia un'eccezione.
- **`void post_delayed(`**
  **`std::shared_ptr< std::packaged_task<void()> > pt,`**
  **`std::chrono::milliseconds delay);`**
  Come nel caso precedente inserisce il task nella coda dei messaggi. In questo caso, però, il messaggio non potrà essere estratto prima che scadano delay millisecondi. Eventuali altri messaggi inseriti successivamente nella coda tramite postTask(...) o postDelayed(...) potranno essere elaborati prima del messaggio corrente se la loro scadenza risulterà più ravvicinata. Se questo metodo viene invocato dopo shutdown(), lancia un'eccezione.

Si implementi la classe message_loop sopra descritta utilizzando le funzionalità offerte dalla libreria c++ 2011.

## Esempio di utilizzo

```
int main() {
  message_loop ml;
  ml.startup();
  auto t1= std::make_shared< task >([](){
                             std::cout<<"second message\n";
                             throw 1; });
  auto t3= std::make_shared< task >([](){std::cout<<"third message\n";});
  auto t2= std::make_shared< task >([&ml,t3](){
        std::cout<<"first message\n";
        ml.post_delayed(t3, std::chrono::milliseconds(1500)); });
  ml.post_delayed(t1 , std::chrono::milliseconds(1000));
  ml.post_message(t2);
  auto f1= t1->get_future();
  try {
    f1.get();
  } catch (...) {
    std::cout<<"Exception caught in t1\n";
  }
  std::this_thread::sleep_for(std::chrono::milliseconds(5000));
  ml.shutdown();
}
```

## Competenze da acquisire

- Tecniche di sincronizzazione
- Uso di packaged_task, condition_variable, once_flag
- Uso delle funzioni relative allo scorrere del tempo