```
#ifndef TIMER_H
#define TIMER_H

#include <thread>
#include <chrono>
#include <functional>
#include <future>
#include <cstdio>
#include <iostream>

class Timer
{
    public:

        Timer(){};

        template <class callable>
        //Recibe como parámetro la función f, la cual es de tipo
       //callable lo que quiere decir que puede ser invocada con la
       //función 'invoke'. f es una referencia a un rvalue
        void connect(callable&& f)
        {
            std::thread([=]()
            {
                while(true)
                {
                    if(go.load()) //Equivalente a 'if (go)' pero
                                  // realizada de forma atómica.

                       //En vez de emitir un timeout y que eso active
                       //la funcion cuenta, llama a la función
                       // directamente
                        std::invoke(f); //Ejecuta la función f pasada
                                        // como puntero

                   //Duerme el hilo los milisegundos que contenga
                   // 'period'
std::this_thread::sleep_for(std::chrono::milliseconds(period.load()));
                }
            }).detach(); //Separa este hilo del principal para que
                         //continue su ejecución de forma
                         //independiente
        };

        void start(int p)
        {
                       //Estas sentencias son equivalentes a
                       //'period = p' y 'go = true' (el valor entre
                       //parentesis se copia al objeto que llama)
                       //pero, al ser 'period' y 'go' tipos atomicos,
                        //es necesario usar una funcion atomica
                       period.store(p);
                       go.store(true);
        };

        void stop() { go.store(!go); };

        void setPeriod(int p) { period.store(p) ;};
```

```
    private:
        //'go' y 'period' son objetos de tipos atomicos. Estos objetos
        //se usan para evitar condiciones de carrera
        // en las que dos hilos acceden al valor de una variable al
        //mismo tiempo y al ejecutar operaciones sobre esa
        // variable uno de los hilos, el valor no es correcto si el
        //otro ya la ha modificado
        std::atomic_bool go = false;
        std::atomic_int period = 0;


};

#endif // TIMER_H
```