```
#include <chrono>
#include <iostream>
#include <thread>
#include <vector>
#include <numeric>

//D'oh! A "using" directive? WTF!
using namespace std::chrono;

//Determine our platform's tic period
const double microsPerClkTic{
  1.0E6 * system_clock::period::num / system_clock::period::den };

//Our REQUIRED processing period
const milliseconds intervalPeriodMillis{ 10 };

//The number of periods we'll run our test fixture for
const int32_t numLoops{ 3000 + 1 };

void periodicTask() {

  std::cout << "system_clock precision = "
            << microsPerClkTic
            << " microseconds/tic"
            << std::endl
            << "Desired Wakeup Period = "
            << intervalPeriodMillis.count()
            << " milliseconds"
            << std::endl
            << "Num Wakeups = "
            << numLoops - 1
            << std::endl;

  //Initialize the chrono timepoint & duration objects we'll be
  //using over & over inside our sleep loop
  system_clock::time_point currentStartTime{ system_clock::now() };
  system_clock::time_point nextStartTime{ currentStartTime };

  //For storing our errors
  std::vector<system_clock::duration> wakeupErrors{};
  wakeupErrors.reserve(numLoops);

  int32_t loopNum{}; //loop counter

  //Let's ROCK!!!!!!
  while (loopNum < numLoops) {

    //Get our current "wakeup" time
    currentStartTime = system_clock::now();

    //Store the wakeup error for this pass
    wakeupErrors.push_back(currentStartTime - nextStartTime);

    //Determine the point in time at which we want to wakeup for the
```

```
    //next pass through the loop.
    nextStartTime =
      currentStartTime + intervalPeriodMillis;

    //Sleep till our next period start time
    std::this_thread::sleep_until(nextStartTime);

    ++loopNum;

  } //end while

  //Print the average wakeup error
  system_clock::duration sum{};
  sum = std::accumulate(std::begin(wakeupErrors) + 1,
                        std::end(wakeupErrors),
                        sum);

  std::cout << "\nAverage Wakeup Error = "
            << duration_cast<microseconds>(sum).count() / (numLoops - 1)
            << " microseconds\n";

}

int main() {

  periodicTask();

  std::cout << "\nPress a key and then press Enter"
            << std::endl;

  char ch{};
  std::cin >> ch;

}
```