Subject: High resolution TimeStop code
Posted by cbpporter on Thu, 05 Mar 2015 10:30:37 GMT
View Forum Message <> Reply to Message

As mentioned in another thread, I require high resolution timers. Most things that one need to measure require precise measurements and TimeStop has a minimum time increment of 0/15/16 msec.

Here is a code I rewrote (I keep loosing ti each time I change U++ version) for TimeStopHR, which has adequate but not maximum precision:

```
class TimeStopHR : Moveable<TimeStopHR> {
 LARGE_INTEGER start;
  LARGE_INTEGER stop;
  LARGE_INTEGER freq;

public:
 double  Elapsed() {
  QueryPerformanceCounter(&stop);
  return (stop.QuadPart - start.QuadPart) * 1000.0 / freq.QuadPart;
 }

 double Seconds()                { return (double)Elapsed() / 1000; }
 String ToString();
 void   Reset();

 TimeStopHR();

 static void SetProcessorAffinity();
};



void TimeStopHR::Reset()
{
 QueryPerformanceFrequency(&freq);
 QueryPerformanceCounter(&start);
}

TimeStopHR::TimeStopHR()
{
 Reset();
}

String TimeStopHR::ToString()
{
 double time = Elapsed();
```

```
 return Format("%d.%03d", int(time / 1000), int(time) % 1000);
}

void TimeStopHR::SetProcessorAffinity()
{
    // Assign the current thread to one processor. This ensures that timing
    // code runs on only one processor, and will not suffer any ill effects
    // from power management.
    //
    // Based on the DXUTSetProcessorAffinity() function in the DXUT framework.

    DWORD_PTR dwProcessAffinityMask = 0;
    DWORD_PTR dwSystemAffinityMask = 0;
    HANDLE hCurrentProcess = GetCurrentProcess();

    if (!GetProcessAffinityMask(hCurrentProcess, &dwProcessAffinityMask,
&dwSystemAffinityMask))
        return;

    if (dwProcessAffinityMask)
    {
        // Find the lowest processor that our process is allowed to run against.

        DWORD_PTR dwAffinityMask = (dwProcessAffinityMask & ((~dwProcessAffinityMask) + 1));

        // Set this as the processor that our thread must always run against.
        // This must be a subset of the process affinity mask.

        HANDLE hCurrentThread = GetCurrentThread();

        if (hCurrentThread != INVALID_HANDLE_VALUE)
        {
            SetThreadAffinityMask(hCurrentThread, dwAffinityMask);
            CloseHandle(hCurrentThread);
        }
    }

    CloseHandle(hCurrentProcess);
}
```

SetProcessorAffinity is optional and ensures even more added precision.

Could we get this added to core as a new class or a replacement for the GetTickCount based TimeStop?

Subject: Re: High resolution TimeStop code
Posted by mirek on Thu, 05 Mar 2015 11:40:51 GMT
View Forum Message <> Reply to Message

I would gladly replace/add, but I need POSIX version too...

Subject: Re: High resolution TimeStop code
Posted by Didier on Fri, 06 Mar 2015 17:19:30 GMT
View Forum Message <> Reply to Message

Hi Cbporter and Mirek,

Here is what I use when need very precise timings, it works for Win and Linux (notice the 'timeType': that is intended to be used by client classes in order make the code portable accross OS.

HWTiming is the 'Timer Facility' type that must be used in code.

HwTiming.h
```
#ifdef WIN32
 #include <Windows.h>
 class windowsHWTiming
 {
  protected:
   windowsHWTiming(void);

  public:
   typedef struct
   {
     signed __int64 nTicksCnt;
   } timeType;

   inline double diff_ms(timeType& p_start, timeType& p_end)
   {
    return (double(p_end.nTicksCnt-p_start.nTicksCnt)/double(m_TickPerSecond)*1000.);
   };

   inline timeType getTime(void)
   {
    timeType res;
    QueryPerformanceCounter((LARGE_INTEGER*)&res.nTicksCnt);
    return res;
   };

  private:
   signed __int64 m_TickPerSecond;
 };
```

```
 typedef windowsHWTiming HWTiming;

#else
 #include <time.h>
 class LinuxHWTiming {
  protected:
   LinuxHWTiming() {};

  public:
   typedef timespec timeType;

   // prend le temps exact
   static inline timeType getTime()
   {
    timeType t;
    clock_gettime(CLOCK_REALTIME, &t);
    return t;
   }

   // renvoie la difference en ms (milli-secondes)
   static inline double diff_ms(timeType& t1, timeType& t2)
   {
    return ((t2.tv_sec-t1.tv_sec)*1000.0 + (t2.tv_nsec-t1.tv_nsec)/1000000.0);
   }
 };

 typedef LinuxHWTiming HWTiming;
#endif


HwTiming.cpp

#include "HWTiming.h"

#ifdef WIN32

  windowsHWTiming::windowsHWTiming(void)
  {
    QueryPerformanceFrequency((LARGE_INTEGER*)&m_TickPerSeco nd);
  };
#endif
```

Subject: Re: High resolution TimeStop code
Posted by cbpporter on Tue, 23 Jun 2015 16:20:02 GMT

I did some testing and at least for relatively short tasks where precision is required, SetProcessorAffinity does not seem to be needed.

Didier's version is under Windows virtually the same as mine, so the Linux version could be used to produce a POSIX implementation too. On the other hand, I do not know the GetTickCount you implement under Linux and its resolution, so there might be no need for an improvement there.

Subject: Re: High resolution TimeStop code
Posted by Didier on Tue, 23 Jun 2015 17:39:25 GMT

Hi cbporter,

The windows version doesn't need the porcessor affinity function you set, read the folowing article :
https://msdn.microsoft.com/en-us/library/windows/desktop/dn5 53408%28v=vs.85%29.aspx

The linux version doesn't need one, the CLOCK_REALTIME parameter is sufficient.

Subject: Re: High resolution TimeStop code
Posted by cbpporter on Tue, 03 Jan 2017 12:10:38 GMT

Since high resolution timer issue was not solved I switched away from TimeStop a long time ago.

I use StopWatch right now. Been using it for over a year, so I hope it is good. I'll try to merge it into TimeStop.

BTW, I'm not a native English speaker and I needed a native English speaker to tell me "WTF is a TimeStop? You mean a StopWatch?".

The answer was yes...

```
#ifndef __TIMESTOP_HPP__
#define __TIMESTOP_HPP__

#include <Core/Core.h>

using namespace Upp;

#ifdef PLATFORM_WIN32
```

```
class StopWatch : Moveable<StopWatch> {
 LARGE_INTEGER start;
 LARGE_INTEGER stop;
 LARGE_INTEGER freq;

public:
 double  Elapsed() {
  QueryPerformanceCounter(&stop);
  return (stop.QuadPart - start.QuadPart) * 1000.0 / freq.QuadPart;
 }

 double Seconds() {
  return (double)Elapsed() / 1000;
 }

 String ToString() {
  double time = Elapsed();
  return Format("%d.%03d", int(time / 1000), int(time) % 1000);
 }

 void Reset() {
  QueryPerformanceFrequency(&freq);
  QueryPerformanceCounter(&start);
 }

 StopWatch() {
  Reset();
 }
};

#endif

#ifdef PLATFORM_POSIX

#include <time.h>

class StopWatch : Moveable<StopWatch> {
 timespec start;

public:
 double  Elapsed() {
  timespec end;
  clock_gettime(CLOCK_REALTIME, &end);

  return (end.tv_sec - start.tv_sec) * 1000.0 + (end.tv_nsec - start.tv_nsec) / 1000000.0;
 }
```

```
 double Seconds() {
  return (double)Elapsed() / 1000;
 }

 String ToString() {
  double time = Elapsed();
  return Format("%d.%03d", int(time / 1000), int(time) % 1000);
 }

 void Reset() {
  clock_gettime(CLOCK_REALTIME, &start);
 }

 StopWatch() {
  Reset();
 }
};

#endif

#endif
```