Subject: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by Wolfgang on Wed, 09 May 2012 10:17:36 GMT
View Forum Message <> Reply to Message

I want to know how long (in ms) it takes to get a response from my server through a udp connection. (from a ping)

header:
Time lastPingRequest;

send method:
```
bool hcan::doSend(uint16 what, CanFrame& frame)
{
  if (!service.prepFrame(what, frame))
    return false;
  if (what == CanMethod::PING_REQUEST)
    lastPingRequest = GetSysTime();

  if (urc.RawSend(frame))
  ...
}
```

receive method:
```
case CanMethod::PING_REPLAY:
{
  int64 pingLatency = GetSysTime() - ToTime(lastPingRequest);
  String tLogMsg = "Ping latency: ";
  tLogMsg << AsString(pingLatency);
  Log(2,tLogMsg);
}
```

But this returns me something like:
Ping latency: 43909
Ping latency: 44138

I think I have more than one fault in my try to get the time... but I found nothing that I could understand to RTIMING etc.

Can someone give me a small example how to do?

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by dolik.rce on Wed, 09 May 2012 11:34:11 GMT
View Forum Message <> Reply to Message

Hi Wolfgang,

GetSysTime() returns Time, which works with resolution in seconds. That is not really helpful if you want to track network actions that happen in milliseconds.

What you want to use is GetTickCount() which returns number of milliseconds since epoch. Unfortunatelly, it returns dword value, which means it overflows every 49.7 days. So you might get weird results every now and then  On windows this can be fixed by using GetTickCount64(). On other platforms, there is no such function in U++ right now.

Anyway, GetTickCount() should be fine for your purpose. Here is simple example: #include <Core/Core.h>
using namespace Upp;

```
dword start;

void func_a(){
 start = GetTickCount();
};

void func_b(){
 LOG("Time: " << (GetTickCount()-start) << " ms");
};

CONSOLE_APP_MAIN{
 func_a();
 Sleep(1234);
 func_b();
}
```

Just for completeness: *TIMING() macros do something slightly different. They measure how long and how often some block of code is executed. E.g. RTIMING("some_label"){  DoSomething(); } will measure how long it took to execute DoSomething() and how many times it was executed from this exact place. The results are then written in log, together with some statistics.

Best regards,
Honza

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by Wolfgang on Wed, 09 May 2012 11:44:43 GMT
View Forum Message <> Reply to Message

Okay, thank you.
Hadn't thought that it is that easy!

The "problem" with overflowing every 2^32ms isn't really a problem for me with time periods below 1000ms.

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by cbpporter on Wed, 09 May 2012 13:26:09 GMT
View Forum Message <> Reply to Message

The real problem with GetTickCount is the resolution. I find it way too low for most practical purposes, because two close measurements will almost always return 0, 15 or 16 ms.

I don't have a proper solution at hand right now, but this should work fairly well:

```
class HRTimer {
private:
   LARGE_INTEGER start;

   LARGE_INTEGER stop;

   double frequency;

public:
   HRTimer() {
      frequency = GetFrequency();
   }

   double GetFrequency();
   void StartTimer();
   double StopTimer();
};

double HRTimer::GetFrequency(void) {
   LARGE_INTEGER proc_freq;

   if (!::QueryPerformanceFrequency(&proc_freq)) {
      ASSERT(0);
   }

   return proc_freq.QuadPart;
}

void HRTimer::StartTimer(void) {
   DWORD_PTR oldmask = ::SetThreadAffinityMask(::GetCurrentThread(), 0);

   ::QueryPerformanceCounter(&start);

   ::SetThreadAffinityMask(::GetCurrentThread(), oldmask);
}

double HRTimer::StopTimer(void) {
   DWORD_PTR oldmask = ::SetThreadAffinityMask(::GetCurrentThread(), 0);

   ::QueryPerformanceCounter(&stop);
```

```
    ::SetThreadAffinityMask(::GetCurrentThread(), oldmask);

    return ((stop.QuadPart - start.QuadPart) / frequency);
}
```

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by Wolfgang on Wed, 09 May 2012 13:48:56 GMT
View Forum Message <> Reply to Message

hmm, looks interessting.
Isn't this a little bit to time consuming?

I'll try later if this would do the job better for me but with the "simple" GetTickCount I get values between 2 and ~60ms for a ping packet from my app to my local server - and this seems to be ok, for me?!

But thanks anyway for your answer, its quite nice to see another way with a higher resolution. Maybe I'll need it in the future..

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by dolik.rce on Wed, 09 May 2012 16:07:13 GMT
View Forum Message <> Reply to Message

You are right cbporter, for very short times it is not precise enough. Also it can depend on the way compiler optimizes the code, I think. If one just needs to now if it takes 10ms or 100ms to send a packet, it is accurate enough

IMHO the best way to achieve good resolution is usually to perform the task many times and measure total time, averaging it later - that is the way TIMING() does it. Using this trick you can get resolution in microseconds, assuming there is not too big deviation between the lengths of single actions.

Honza

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by Didier on Wed, 09 May 2012 20:23:11 GMT
View Forum Message <> Reply to Message

I use this to measure timings, it works in linux and windows.

Timing.h

```
#ifdef WIN32
```

```
#include <Windows.h>
class windowsTiming
{
 public:
  windowsTiming(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 windowsTiming HWTiming;

#else
 #include <time.h>
 class LinuxTiming {
  public:
   LinuxTiming() {};

  public:
   typedef timespec timeType;

   static inline timeType getTime()
   {
    timeType t;
    clock_gettime(CLOCK_REALTIME, &t);
    return t;
   }

   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 LinuxTiming HWTiming;
#endif
```

Timing.c

```
#ifdef WIN32
  windowsTiming::windowsTiming(void)
  {
   QueryPerformanceFrequency((LARGE_INTEGER*)&m_TickPerSecond);
  };
#endif
```

To use it in a cross plateform way, do the following:

```
HWTiming timeMeasurer;

HWTiming::timeType  startTime = timeMeasurer.getTime();

.... do you're thing !-)

HWTiming::timeType  endTime = timeMeasurer.getTime();

double delta = timeMeasurer.diff_ms(endTime, startTime);
```

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by dolik.rce on Fri, 11 May 2012 18:23:24 GMT
View Forum Message <> Reply to Message

I just accidentally stumbled upon the new C++11 way to measure times. With <chrono> header, you can do something like this:

```
#include <chrono> //requires -std=c++11

#include <Core/Core.h>
using namespace Upp;

CONSOLE_APP_MAIN{
 auto start = std::chrono::high_resolution_clock::now();
 Sleep(1); //execute your code here
 auto duration = std::chrono::high_resolution_clock::now() - start;
```

```
 auto result=std::chrono::duration_cast<std::chrono::microseconds>(duration).count();
 LOG(result << " us");
}
```

Sorry for the stupid use of the "auto" keyword... I was too lazy to type the full type names for time_point and duration - they are almost as long as in Java  Reference for the header can be found at this great site: http://en.cppreference.com/w/cpp/chrono (thanks Sender Ghost for showing me  )

The resolution seems to be in microseconds (when I formatted the output to nanoseconds, it showed only multiples of 1000) on my system, but it might vary on other machines/platforms.

Best regards,
Honza

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by cbpporter on Sat, 12 May 2012 08:13:03 GMT
View Forum Message <> Reply to Message

std::chrono::duration_cast<std::chrono::microseconds>(duration).count()

Really? WTF is happening with C++?

---

Subject: Re: time measurement :: RTIMING, TimeStop, GetTickCount
Posted by dolik.rce on Sat, 12 May 2012 10:03:48 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sat, 12 May 2012 10:13std::chrono::duration_cast<std::chrono::microseconds>(duration).count()

Really? WTF is happening with C++?
I thought exactly the same when I saw it first  It would get better with "using namespace std::chrono;", but still the style is not really up to my taste - a lot of static methods, specialized type members in each class, and other syntactic constructions that just make it all longer... Maybe we should create a nice, shorthand U++ wrapper

Honza

---