Subject: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Fri, 07 Jan 2011 15:38:50 GMT
View Forum Message <> Reply to Message

Hello All,

I propose to add new static methods to Thread class:

```
class Thread : NoCopy {
public:
#ifdef PLATFORM_WIN32
 typedef HANDLE HandleType;
 typedef dword IdType;
#elif defined(PLATFORM_POSIX)
 typedef pthread_t HandleType;
 typedef pthread_t IdType;
#else
 #error Thread is not support for this platform
#endif
private:
 HandleType     handle;
....
public:
 static IdType GetCurrentThreadId();
 static HandleType GetCurrentThreadHandle();
...
```

and cpp source c ode:

```
Thread::IdType Thread::GetCurrentThreadId(){
#ifdef PLATFORM_WIN32
 return GetCurrentThreadId();
#elif defined(PLATFORM_POSIX)
 return pthread_self();
#else
 #error Thread is not support for this platform
#endif
}

Thread::HandleType Thread::GetCurrentThreadHandle(){
#ifdef PLATFORM_WIN32
 return GetCurrentThread();
#elif defined(PLATFORM_POSIX)
 return pthread_self();
#else
```

```
 #error Thread is not support for this platform
#endif
}
```

This method help me to identify the current thread id.

ADD: new method: Thread::GetCurrentThreadHandle

Best regards,
Ion

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Sat, 08 Jan 2011 12:07:26 GMT
View Forum Message <> Reply to Message

Ok, added this:

```
class Thread : NoCopy {
#ifdef PLATFORM_WIN32
 HANDLE     handle;
#endif
#ifdef PLATFORM_POSIX
 pthread_t  handle;
#endif

public:
 bool       Run(Callback cb);

 void       Detach();
 int        Wait();

 bool       IsOpen() const     { return handle; }


#ifdef PLATFORM_WIN32
 typedef HANDLE Handle;
#endif
#ifdef PLATFORM_POSIX
 typedef pthread_t Handle;
#endif

 Handle      GetHandle() const { return handle; }
```

```
 void        Priority(int percent); // 0 = lowest, 100 = normal

 static void Start(Callback cb);

 static void Sleep(int ms);

 static bool IsST();
 static bool IsMain();
 static int  GetCount();
 static void ShutdownThreads();
 static bool IsShutdownThreads();

#ifdef PLATFORM_WIN32
 static Handle GetCurrentHandle()    { return GetCurrentThread(); }
#endif
#ifdef PLATFORM_POSIX
 static Handle GetCurrentHandle()    { return pthread_self(); }
#endif

 Thread();
 ~Thread();

private:
 void operator=(const Thread&);
 Thread(const Thread&);
};
```

ThreadId does not make sense for me now (IMO: it is Win32 specific and not really related to Thread).

Note: The faster alternative to all this might be checking the pointer to TLS variable.

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Sun, 09 Jan 2011 21:28:29 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 08 January 2011 14:07Ok, added this:

ThreadId does not make sense for me now (IMO: it is Win32 specific and not really related to Thread).

Mirek,

I think that you are not right according by:

http://suacommunity.com/dictionary/pthread_self-entry.php

Quote:In the Windows threading model each created thread has both a HANDLE and a system-wide unique id. As a result the GetCurrentThreadId Windows function returns the same logical information as the POSIX call.

The method DWORD WINAPI GetCurrentThread(void) returns the pseudo-handle, that is not same with the result _beginthreadex(...).

In the other had, you are right, because in POSIX you can manage with the result pthread_self.

In the end, I need an unique Thread ID.

mirek wrote on Sat, 08 January 2011 14:07Ok, added this:

ThreadId does not make sense for me now (IMO: it is Win32 specific and not really related to Thread).

Note: The faster alternative to all this might be checking the pointer to TLS variable.
About Your Note, can you give me an example, please?

Thank you in advance!

Added:
By TLS variable you mean: Thread-local storage variable?

Like this:

```
thread__ bool sThreadId;

qword GetCurrentThreadIdCustom(){
 return (qword (&sThreadId));
}
```

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Sun, 09 Jan 2011 21:49:57 GMT
View Forum Message <> Reply to Message

tojocky wrote on Sun, 09 January 2011 16:28mirek wrote on Sat, 08 January 2011 14:07Ok, added this:

ThreadId does not make sense for me now (IMO: it is Win32 specific and not really related to Thread).

Mirek,

I think that you are not right according by:
http://suacommunity.com/dictionary/pthread_self-entry.php

Quote:In the Windows threading model each created thread has both a HANDLE and a system-wide unique id. As a result the GetCurrentThreadId Windows function returns the same logical information as the POSIX call.

The method DWORD WINAPI GetCurrentThread(void) returns the pseudo-handle, that is not same with the result _beginthreadex(...).

In the other had, you are right, because in POSIX you can manage with the result pthread_self.

In the end, I need an unique Thread ID.

Well, true, but we should find a better method how to integrate it..

Quote:
mirek wrote on Sat, 08 January 2011 14:07Ok, added this:

ThreadId does not make sense for me now (IMO: it is Win32 specific and not really related to Thread).

Note: The faster alternative to all this might be checking the pointer to TLS variable.
About Your Note, can you give me an example, please?

Thank you in advance!

Added:
By TLS variable you mean: Thread-local storage variable?

Like this:

```
thread__ bool sThreadId;

qword GetCurrentThreadIdCustom(){
 return (qword (&sThreadId));
}
```

Basically yes. If inlined, it should translate into simple [fs] based load CPU op... (no calls to API).

Mirek

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Mon, 10 Jan 2011 14:35:40 GMT
View Forum Message <> Reply to Message

Ok, Exists 2 variants:
1. The first variant, proposed by me, which calls system API:

```
class Thread : NoCopy {
#ifdef PLATFORM_WIN32
 HANDLE     handle;
#endif
#ifdef PLATFORM_POSIX
 pthread_t  handle;
#endif
public:
 bool       Run(Callback cb);

 void       Detach();
 int        Wait();

 bool       IsOpen() const     { return handle; }

#ifdef PLATFORM_WIN32
 typedef HANDLE Handle;
 typedef DWORD Id;
#endif
#ifdef PLATFORM_POSIX
 typedef pthread_t Handle;
 typedef pthread_t Id;
#endif
 Handle      GetHandle() const { return handle; }

 void        Priority(int percent); // 0 = lowest, 100 = normal

 static void Start(Callback cb);

 static void Sleep(int ms);

 static bool IsST();
 static bool IsMain();
 static int  GetCount();
 static void ShutdownThreads();
 static bool IsShutdownThreads();
#ifdef PLATFORM_WIN32
 static Handle GetCurrentHandle(){
  return GetCurrentThread();
 }
 static Id GetCurrentId(){
  return ::GetCurrentThreadId();
```

```
 };
 static inline
#elif defined(PLATFORM_POSIX)
 static Handle GetCurrentHandle(){
  return pthread_self();
 }
 static Id GetCurrentId(){
  return pthread_self();
 };
#endif


 Thread();
 ~Thread();

private:
 void operator=(const Thread&);
 Thread(const Thread&);
};
```

2. The second variant, customized, proposed by Mirek, which need to be faster than call system API is:

```
static thread__ bool __sThreadId;
...
class Thread : NoCopy {
#ifdef PLATFORM_WIN32
 HANDLE     handle;
#endif
#ifdef PLATFORM_POSIX
 pthread_t  handle;
#endif
public:
 bool       Run(Callback cb);

 void       Detach();
 int        Wait();

 bool       IsOpen() const     { return handle; }

#ifdef PLATFORM_WIN32
 typedef HANDLE Handle;
#endif
#ifdef PLATFORM_POSIX
 typedef pthread_t Handle;
#endif
 typedef qword Id;
```

```
 Handle      GetHandle() const { return handle; }

 void        Priority(int percent); // 0 = lowest, 100 = normal

 static void Start(Callback cb);

 static void Sleep(int ms);

 static bool IsST();
 static bool IsMain();
 static int  GetCount();
 static void ShutdownThreads();
 static bool IsShutdownThreads();
#ifdef PLATFORM_WIN32
 static Handle GetCurrentHandle(){
  return GetCurrentThread();
 }
#elif defined(PLATFORM_POSIX)
 static Handle GetCurrentHandle(){
  return pthread_self();
 }
#endif
 static inline Id GetCurrentId(){
  return (qword)(&__sThreadId);
 };


 Thread();
 ~Thread();

private:
 void operator=(const Thread&);
 Thread(const Thread&);
};
```

The "static thread__ bool __sThreadId" and "GetCurrentId()" menthod body in the second variant can be placed in cpp file.

What is the best solution?

Thank you Mirek for Hint!

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by dolik.rce on Mon, 10 Jan 2011 15:25:25 GMT
View Forum Message <> Reply to Message

Maybe there should be both methods. There are still some compilers without TLS support (such

as the non-TDM mingw) and some users are not really aware of this fact. The faster solution could be default, with #ifdefed fallback to the API call for the compilers without the TLS support.

Honza

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Mon, 10 Jan 2011 16:48:46 GMT
View Forum Message <> Reply to Message

Should not we have GetId() (instance specific) too?

Mirek

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Mon, 10 Jan 2011 18:13:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 10 January 2011 18:48Should not we have GetId() (instance specific) too?

Mirek

Sorry I didn't undesrtood what did you mean!

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Tue, 11 Jan 2011 13:49:15 GMT
View Forum Message <> Reply to Message

We have

```
Handle      GetHandle() const { return handle; }
```

but no equivalent for thread ID (whereas we have both as static method).

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Tue, 11 Jan 2011 15:11:20 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 11 January 2011 15:49We have

```
 Handle      GetHandle() const { return handle; }
```

but no equivalent for thread ID (whereas we have both as static method).

GetHandle is not a static method. GetHandle and GetCurrentHandle are different methods.

Other Idea:

To demonstrate that Mirek proposed (customized) a more faster than my proposed (API) I have created a simple test running in threads:

```
void WorkerThread(DivisorsInfo f)
{
 char _l[1000];
 Thread::Id id_api;
 Thread::IdCustom id_custom;

 TimeStop v_timer;

 v_timer.Reset();
 for(int i=0;i<100000000;++i){
  id_custom = Thread::GetCurrentIdCustom();
 }
 dword v_time_cust = v_timer.Elapsed();

 v_timer.Reset();
 for(int i=0;i<100000000;++i){
  id_api = Thread::GetCurrentId();
 }
 dword v_time_api1 = v_timer.Elapsed();

 sprintf(_l, "Run Thread: %X. api time %u, custom time %u", Thread::GetCurrentIdCustom(),
v_time_api1, v_time_cust);
}
```

and restul under linux (Ubuntu):

```
Run Thread: B34EEF34. api time 0.991, custom time 1.088. custom realization are faster
0.910846 times
Run Thread: B2CEBF34. api time 1.455, custom time 1.080. custom realization are faster
1.347222 times
Run Thread: B24E8F34. api time 1.862, custom time 0.680. custom realization are faster
2.738235 times
Run Thread: B1CD4F34. api time 2.133, custom time 0.927. custom realization are faster
2.300971 times
Run Thread: B0CAFF34. api time 2.223, custom time 0.698. custom realization are faster
```

3.184814 times
Run Thread: B14C3F34. api time 2.234, custom time 0.903. custom realization are faster 2.473976 times
Run Thread: B049BF34. api time 2.192, custom time 0.648. custom realization are faster 3.382716 times
Run Thread: AFC87F34. api time 2.030, custom time 0.573. custom realization are faster 3.542757 times

and same code under windows xp
Run Thread: 162229. api time 0.672, custom time 0.390. custom realization are faster 1.723077 times
Run Thread: 167561. api time 0.860, custom time 0.984. custom realization are faster 0.873984 times
Run Thread: 162229. api time 1.219, custom time 1.156. custom realization are faster 1.054498 times
Run Thread: 171BD1. api time 1.265, custom time 0.735. custom realization are faster 1.721088 times
Run Thread: 16C899. api time 1.594, custom time 0.890. custom realization are faster 1.791011 times
Run Thread: 176F09. api time 1.485, custom time 0.578. custom realization are faster 2.569204 times
Run Thread: 17C241. api time 1.485, custom time 0.437. custom realization are faster 3.398169 times
In the end: Mirek realization are approximative 2 times faster than API functionality.
Mirek, can you add your proposed functionality?

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Wed, 12 Jan 2011 08:49:08 GMT
View Forum Message <> Reply to Message

tojocky wrote on Tue, 11 January 2011 10:11mirek wrote on Tue, 11 January 2011 15:49We have

```
Handle      GetHandle() const { return handle; }
```

but no equivalent for thread ID (whereas we have both as static method).

GetHandle is not a static method. GetHandle and GetCurrentHandle are different methods.

I am sorry that I described it still in a confusing way.

I mean we are planning to have:

static Handle GetCurrentHandle(){
static Id GetCurrentId(){

but we only have now

Handle GetHandle();

is not

Id GetId();

missing?

As for TLS method, without such GetId() you definitely do not need to add it as part of interface... You can define TLS variable anywhere you need to do this fast And it is true that system Id has some expected meaning...

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Wed, 12 Jan 2011 11:36:52 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 12 January 2011 10:49
I mean we are planning to have:

static Handle GetCurrentHandle(){
static Id GetCurrentId(){

but we only have now

Handle GetHandle();

is not

Id GetId();

missing?

You are right,

Thank you for explanation.

Finaly, The class Thread should have api callback.

header code:

```
class Thread : NoCopy {
#ifdef PLATFORM_WIN32
 HANDLE     handle;
 DWORD    thread_id;
#endif
#ifdef PLATFORM_POSIX
 pthread_t  handle;
 pthread_t  thread_id;
#endif
public:
 bool       Run(Callback cb);

 void       Detach();
 int        Wait();

 bool       IsOpen() const     { return handle; }

#ifdef PLATFORM_WIN32
 typedef HANDLE Handle;
 typedef DWORD Id;
#endif
#ifdef PLATFORM_POSIX
 typedef pthread_t Handle;
 typedef pthread_t Id;
#endif
 typedef qword IdCustom;
 Handle      GetHandle() const { return handle; }
 Id          GetId() const {return thread_id;}

 void        Priority(int percent); // 0 = lowest, 100 = normal

 static void Start(Callback cb);

 static void Sleep(int ms);
```

```
 static bool IsST();
 static bool IsMain();
 static int  GetCount();
 static void ShutdownThreads();
 static bool IsShutdownThreads();
#ifdef PLATFORM_WIN32
 static Handle GetCurrentHandle(){
  return GetCurrentThread();
 }
 static inline Id GetCurrentId(){
  return ::GetCurrentThreadId();
 };
#elif defined(PLATFORM_POSIX)
 static Handle GetCurrentHandle(){
  return pthread_self();
 }
 static inline Id GetCurrentId(){
  return pthread_self();
 };
#endif

 Thread();
 ~Thread();

private:
 void operator=(const Thread&);
 Thread(const Thread&);
};
```

and cpp code:

```
Thread::Thread()
{
 sMutexLock();
#ifdef PLATFORM_WIN32
 handle = 0;
 thread_id = 0;
#endif
#ifdef PLATFORM_POSIX
 handle = 0;
 thread_id = 0;
#endif
}

void Thread::Detach()
{
```

```
#if defined(PLATFORM_WIN32)
 if(handle) {
  CloseHandle(handle);
  handle = 0;
  thread_id = 0;
 }
#elif defined(PLATFORM_POSIX)
 if(handle) {
  CHECK(!pthread_detach(handle));
  handle = 0;
  thread_id = 0;
 }
#endif
}

bool Thread::Run(Callback _cb)
{
 AtomicInc(sThreadCount);
 if(!threadr)
#ifndef CPU_BLACKFIN
  threadr = sMain = true;
#else
 {
  threadr = true;
  //the sMain replacement
#ifdef PLATFORM_POSIX
  pthread_t thid = pthread_self();
  vm.Enter();
  if(threadsv.Find(thid) < 0){
   //thread not yet present, mark present
   threadsv.Add(thid);
  }
  else
   RLOG("BUG: Multiple Add in Mt.cpp");
  vm.Leave();
#endif
 }
#endif
 Detach();
 Callback *cb = new Callback(_cb);
#ifdef PLATFORM_WIN32
 handle = (HANDLE)_beginthreadex(0, 0, sThreadRoutine, cb, 0, ((unsigned int *)(&thread_id)));
#endif
#ifdef PLATFORM_POSIX
 if(pthread_create(&handle, 0, sThreadRoutine, cb))
  handle = 0;
 thread_id = handle;
#endif
```

```
 return handle;
}
```

mirek wrote on Wed, 12 January 2011 10:49
As for TLS method, without such GetId() you definitely do not need to add it as part of interface... You can define TLS variable anywhere you need to do this fast And it is true that system Id has some expected meaning...

I agree with you!

And Thank you for a good cooperation.

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Wed, 12 Jan 2011 19:40:35 GMT
View Forum Message <> Reply to Message

Commited. I have made a small change in posix, please check...

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Thu, 13 Jan 2011 10:47:32 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 12 January 2011 21:40Commited. I have made a small change in posix, please check...

It is not OK. because, if you set thread_id only for PLATFORM_WIN32 then remove initialization from constructor and detach for POSIX initialization of thread_id = 0;.

```
Thread::Thread()
{
 sMutexLock();
#ifdef PLATFORM_WIN32
 handle = 0;
 thread_id = 0;
#endif
#ifdef PLATFORM_POSIX
 handle = 0;
==>remove this thread_id = 0;
#endif
}
```

```
void Thread::Detach()
{
#if defined(PLATFORM_WIN32)
 if(handle) {
  CloseHandle(handle);
  handle = 0;
  thread_id = 0;
 }
#elif defined(PLATFORM_POSIX)
 if(handle) {
  CHECK(!pthread_detach(handle));
  handle = 0;
==>remove this thread_id = 0;
 }
#endif
}
```

In rest is OK!

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by mirek on Thu, 13 Jan 2011 19:51:51 GMT
View Forum Message <> Reply to Message

Sorry about that... applied.

Mirek

Subject: Re: Thread::GetCurrentThreadId() and Thread::GetCurrentThreadHandle() new methods
Posted by tojocky on Fri, 14 Jan 2011 07:12:21 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 13 January 2011 21:51Sorry about that... applied.

Mirek

Now is OK!

Thank you for team work!
