Subject: SSL server crash
Posted by Zbych on Thu, 08 Sep 2011 14:33:20 GMT
View Forum Message <> Reply to Message

Hi,

I have a problem with a simple SSL server on linux. After connecting a few clients (for example 5), I start killing them one by one. Server should just close connections, but it crashes inside SetSockError:

```
void Socket::SetSockError(SOCKET socket, const char *context, int code, const char *errdesc)
{
 String err;
 errorcode = code;
 if(socket != INVALID_SOCKET)
  err << "socket(" << (int)socket << ") / ";
 err << context << ": " << errdesc;
 errordesc = err; //<------------ crash
 is_error = true;
 SetErrorText(err);
}
```

The question is what I am doing wrong? Maybe I shouldn't use vector in two different threads without protection?

File Attachments

1) SockVect.cpp, downloaded 450 times

Subject: Re: SSL server crash
Posted by mirek on Thu, 08 Sep 2011 18:42:38 GMT
View Forum Message <> Reply to Message

Zbych wrote on Thu, 08 September 2011 10:33Hi,

I have a problem with a simple SSL server on linux. After connecting a few clients (for example 5), I start killing them one by one. Server should just close connections, but it crashes inside SetSockError:

```
void Socket::SetSockError(SOCKET socket, const char *context, int code, const char *errdesc)
{
 String err;
 errorcode = code;
 if(socket != INVALID_SOCKET)
```

```
  err << "socket(" << (int)socket << ") / ";
 err << context << ": " << errdesc;
 errordesc = err; //<---------- crash
 is_error = true;
 SetErrorText(err);
}
```

The question is what I am doing wrong? Maybe I shouldn't use vector in two different threads without protection?

Not sure if that is the cause, but you definitiely should NOT do client.Add in one thread and client.Remove in another, you need mutex serialization for that...

Mirek

Subject: Re: SSL server crash
Posted by Zbych on Fri, 09 Sep 2011 07:11:56 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 08 September 2011 20:42Not sure if that is the cause, but you definitiely should NOT do client.Add in one thread and client.Remove in another, you need mutex serialization for that...

Mirek

More interesting question is whether I can use Add and operator[] (protected by mutex) at the same time in two different threads . Vector manual states that Add "Invalidates iterators and references to Vector."

It seems that answer to my question is RTFM and use Array + mutex

---

Subject: Re: SSL server crash
Posted by mirek on Fri, 09 Sep 2011 08:00:59 GMT
View Forum Message <> Reply to Message

Zbych wrote on Fri, 09 September 2011 03:11mirek wrote on Thu, 08 September 2011 20:42Not sure if that is the cause, but you definitiely should NOT do client.Add in one thread and client.Remove in another, you need mutex serialization for that...

Mirek

More interesting question is whether I can use Add and operator[] (protected by mutex) at the

same time in two different threads . Vector manual states that Add "Invalidates iterators and references to Vector."

If it is protected by mutex, you can.

Of course, what you cannot is to take a reference (pointer) to element and then unlock the mutex and use this reference. But that is sort of same in single-threaded.

Using Mutex is no magic - each time you need to access a shared variable, you need to "lock" it, so that other thread does not change its content during operation...

Subject: Re: SSL server crash
Posted by Zbych on Fri, 09 Sep 2011 09:36:09 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 09 September 2011 10:00
Of course, what you cannot is to take a reference (pointer) to element and then unlock the mutex and use this reference.

But I can do this with Array, can't I?

Can you take a look at this version:

```
#include <Core/Core.h>
#include <Web/SSL/WebSSL.h>

using namespace Upp;

struct client_data: Moveable<client_data> {
 Socket sock;
 dword ip;
 int par1;
 int par2;
};


template <class T>
class MTArray: private Array<T>{
private:
 Mutex mtx;
public:
 T&       Add(const T& x)     {Mutex::Lock __(mtx); Array<T>::Add(x);}
 void     Remove(int i, int count = 1)  {Mutex::Lock __(mtx); Array<T>::Remove(i, count);}
 const T& operator[](int i) const      {Mutex::Lock __(mtx); return Array<T>::Get(i); }
 T&       operator[](int i)           {Mutex::Lock __(mtx); return Array<T>::Get(i); }
```

```
 int      GetCount()       {Mutex::Lock __(mtx); return Array<T>::GetCount(); }
};


class TestServer{
 volatile Atomic stop;
 MTArray<client_data> clients;

 int WaitForInput(int _timeout_ms);
 void Worker();
public:
 typedef TestServer CLASSNAME;
 void Start();
 void Stop()  {AtomicWrite(stop, true);}
 TestServer()  {AtomicWrite(stop, false);}
};


int TestServer::WaitForInput(int timeout_ms)
{
 fd_set set;
 struct timeval tval;
 int max = 0;

 if (clients.GetCount() <= 0) return -1;

 tval.tv_sec = timeout_ms / 1000;
 tval.tv_usec = 1000 * (timeout_ms % 1000);

 FD_ZERO(&set);
 for (int i = 0; i < clients.GetCount(); i++){
  int tmp = clients[i].sock.GetSocket();
  if (tmp > max) max = tmp;
  FD_SET(tmp, &set);
 }

 if (select(max+1, &set, NULL, NULL, &tval) > 0){
  for (int i = 0; i < clients.GetCount(); i++){
   if (FD_ISSET(clients[i].sock.GetSocket(), &set)) return i;
  }
 }

 return -1;
}


void TestServer::Worker()
{
```

```
 while(!Thread::IsShutdownThreads() && !AtomicRead(stop))
 {
  int ci = WaitForInput(1000);
  if (ci >= 0){
   if (!clients[ci].sock.IsOpen() || clients[ci].sock.IsError() || clients[ci].sock.IsEof()){
    Cout() << "Client no " << ci << " (" << FormatIP(clients[ci].ip) << ") closed connection\n";
    clients[ci].sock.Close();
    clients.Remove(ci);
   }else{
    Cout() << "[" << FormatIP(clients[ci].ip) << "] " << clients[ci].sock.Read() << "\n" ;
   }
  }else{
   for (int i = 0; i < clients.GetCount(); i++){
    clients[i].sock.Write(Format("%`: message to client no %d", GetSysTime(), i));
   }
  }
 }
}


void TestServer::Start()
{
 Socket server;
#if 1
 SSLContext context;

 if (!context.Create(SSLv3_server_method())){
  Cout() << "Can not create context\n";
  return;
 }

 if (!context.UseCertificate(LoadFile(ConfigFile("servercert.pem")),
LoadFile(ConfigFile("serverkey.pem")), false)){
  Cout() << "Certificate and key are diffrent!\n";
  return;
 }

 if(!SSLServerSocket(server, context, 11111, true, 5, true)){
#else
 if(!ServerSocket(server, 11111)){
#endif
  Cout() << "Can not start server\n";
  return;
 }

 Thread().Run(THISBACK(Worker));

 Cout() << "Waiting for connections\n";
```

```
 while(!Thread::IsShutdownThreads() && !AtomicRead(stop)){
  client_data new_client;
  if (server.Accept(new_client.sock, &new_client.ip)){
   Cout() << "Client no " << clients.GetCount() << " from " << FormatIP(new_client.ip) << "\n";
   clients.Add(new_client);
  }else{
   Cout() << "+\n";
  }
 }
}


TestServer server;

void Stop(int sig)
{
 Cout() << "\nstopping, please wait...\n";
 server.Stop();
}


CONSOLE_APP_MAIN
{
 signal(SIGINT, Stop);
 server.Start();
}
```

Subject: Re: SSL server crash
Posted by mirek on Sat, 10 Sep 2011 08:16:24 GMT
View Forum Message <> Reply to Message

This might work as long as you have single working thread. Otherwise

if (!clients[ci].sock.IsOpen() || clients[ci].sock.IsError() || clients[ci].sock.IsEof()

you can get "Remove" at '||'.

It is not a very good code in any case.

Array does not change many things as compered to Vector.

The correct code would be something line:

```
class TestServer{
 volatile Atomic stop;
 Vector<client_data> clients;
	Mutex mtx;

 int WaitForInput(int _timeout_ms);
 void Worker();
public:
 typedef TestServer CLASSNAME;
 void Start();
 void Stop()  {AtomicWrite(stop, true);}
 TestServer()  {AtomicWrite(stop, false);}
};


int TestServer::WaitForInput(int timeout_ms)
{
 fd_set set;
 struct timeval tval;
 int max = 0;

 if (clients.GetCount() <= 0) return -1;

 tval.tv_sec = timeout_ms / 1000;
 tval.tv_usec = 1000 * (timeout_ms % 1000);

	mtx.Enter();
 FD_ZERO(&set);
 for (int i = 0; i < clients.GetCount(); i++){
  int tmp = clients[i].sock.GetSocket();
  if (tmp > max) max = tmp;
  FD_SET(tmp, &set);
 }
	mtx.Leave();

 if (select(max+1, &set, NULL, NULL, &tval) > 0){
		mtx.Enter();
  for (int i = 0; i < clients.GetCount(); i++){
   if (FD_ISSET(clients[i].sock.GetSocket(), &set)) return i;
  }

 }
	else
		mtx.Enter();

 return -1;
}
```

```
void TestServer::Worker()
{
 while(!Thread::IsShutdownThreads() && !AtomicRead(stop))
 {
  int ci = WaitForInput(1000);
  if (ci >= 0){
   if (!clients[ci].sock.IsOpen() || clients[ci].sock.IsError() || clients[ci].sock.IsEof()){
    Cout() << "Client no " << ci << " (" << FormatIP(clients[ci].ip) << ") closed connection\n";
    clients[ci].sock.Close();
    clients.Remove(ci);
   }else{
    Cout() << "[" << FormatIP(clients[ci].ip) << "] " << clients[ci].sock.Read() << "\n" ;
   }
  }else{
   for (int i = 0; i < clients.GetCount(); i++){
    clients[i].sock.Write(Format("%`: message to client no %d", GetSysTime(), i));
   }
  }
                mtx.Leave();
 }
}


void TestServer::Start()
{
 Socket server;
#if 1
 SSLContext context;

 if (!context.Create(SSLv3_server_method())){
  Cout() << "Can not create context\n";
  return;
 }

 if (!context.UseCertificate(LoadFile(ConfigFile("servercert.pem")),
LoadFile(ConfigFile("serverkey.pem")), false)){
  Cout() << "Certificate and key are diffrent!\n";
  return;
 }

 if(!SSLServerSocket(server, context, 11111, true, 5, true)){
#else
 if(!ServerSocket(server, 11111)){
#endif
  Cout() << "Can not start server\n";
  return;
```

```
 }

 Thread().Run(THISBACK(Worker));

 Cout() << "Waiting for connections\n";

 while(!Thread::IsShutdownThreads() && !AtomicRead(stop)){
  client_data new_client;
  if (server.Accept(new_client.sock, &new_client.ip)){
   Cout() << "Client no " << clients.GetCount() << " from " << FormatIP(new_client.ip) << "\n";
mtx.Enter();
   clients.Add(new_client);
mtx.Leave();
  }else{
   Cout() << "+\n";
  }
 }
}


TestServer server;

void Stop(int sig)
{
 Cout() << "\nstopping, please wait...\n";
 server.Stop();
}


CONSOLE_APP_MAIN
{
 signal(SIGINT, Stop);
 server.Start();
}
```

I do not quite like the structure, as we now leave WaitForInput with mutex locked, but I believe it is now correct.

Mirek

Subject: Re: SSL server crash
Posted by Zbych on Thu, 20 Oct 2011 09:34:44 GMT
View Forum Message <> Reply to Message

Thank you Mirek for your review, but since I do not plan to remove clients from other threads I will stick with my version.

I also have a few suggestions regarding Web/SSL package, documentation:

1. WebSSL uses blocking IO while making a new connection, so if a server does not respond to a handshake, ssl client hangs.

2. SSL Server also uses blocking IO and waits for the handshake just after a new client has connected. That means that no new connection is accepted until handshake is done. One can easily make DOS attack on Web/SSL server using plain old telnet client.
I think that SSL handshake should be separated from connection acceptance.

3. SSL_shutdown sends close_notify to the other side, so when it returns 0 it should be called again.

4. When application sends close notify and there is something wrong with the connection, it can receive SIGPIPE. Application should install SIGPIPE handler.

So far to workaround blocking IO, I uncommented timeout functions in socket.cpp and socket.h:

```
  void              WriteTimeout(int msecs);
  void              ReadTimeout(int msecs);
```

```
void Socket::Data::WriteTimeout(int msecs)
{
 ASSERT(IsOpen());
 if(IsNull(msecs)) msecs = 0;
#if defined(PLATFORM_WIN32)
 if(setsockopt(socket, SOL_SOCKET, SO_SNDTIMEO, (const char *)&msecs, sizeof(msecs))) {
  SetSockError("setsockopt(SO_SNDTIMEO)");
 }
#elif defined(PLATFORM_POSIX)
 struct timeval tv;
 tv.tv_sec = msecs / 1000;
 tv.tv_usec = (msecs % 1000) * 1000;
 if(setsockopt(socket, SOL_SOCKET, SO_SNDTIMEO, &tv, sizeof(tv)) < 0) {
  SetSockError("setsockopt(SO_SNDTIMEO)");
 }
#endif
}
```

```
void Socket::Data::ReadTimeout(int msecs)
{
 ASSERT(IsOpen());
 if(IsNull(msecs)) msecs = 0;
#if defined(PLATFORM_WIN32)
 if(setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, (const char *)&msecs, sizeof(msecs))) {
  SetSockError("setsockopt(SO_RCVTIMEO)");
 }
#elif defined(PLATFORM_POSIX)
 struct timeval tv;
 tv.tv_sec = msecs / 1000;
 tv.tv_usec = (msecs % 1000) * 1000;
 if(setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0) {
  SetSockError("setsockopt(SO_RCVTIMEO)");
 }
#endif
}
```

and I added them to SSLSocketData::OpenClient and between calling SSLServerSocket and SSLSocketData::Accept. I did not test timeouts on windows, but on linux they work fine.

```
bool SSLSocketData::OpenClient(const char *host, int port, bool nodelay, dword *my_addr, int timeout, bool blocking)
{

 if(!Data::OpenClient(host, port, nodelay, my_addr, timeout, /*blocking*/true))
  return false;

 Data::ReadTimeout(timeout);
 Data::WriteTimeout(timeout);

 if(!(ssl = SSL_new(ssl_context)))
 {
  SetSSLError("OpenClient / SSL_new");
  return false;
 }
[...]

bool SSLSocketData::Close(int timeout_msec)
{
 if(ssl){
  if(!SSL_shutdown(ssl)){
   Data::StopWrite();
   SSL_shutdown(ssl);
  }
```

```
 }

 bool res = Data::Close(timeout_msec);
 if(ssl) {
  SSL_free(ssl);
  ssl = NULL;
 }
 return res;
}
```

Maybe there is better place to insert those timeouts (for example socket.cpp?). Any suggestions?
