Subject: [SOLVED]Use multiple HttpRequest instance (>32) to upload files will stuck
Posted by kasome on Tue, 29 Jul 2014 03:21:47 GMT
View Forum Message <> Reply to Message

Hi,

I found that when using multiple thread (>32) and each thread has one HttpRequest instance to upload files will lead to stuck in its member function HttpRequest::Dns,

the reason is as follows:

In upp\uppsrc\Core\Inet.h

```
class IpAddrInfo {
 enum { COUNT = 32 };
 struct Entry {
  const char *host;
  const char *port;
  int         family;
  int         status;
  addrinfo   *addr;
 };
 static Entry     pool[COUNT];

 ...............................
};
```

The size of variable IpAddrInfo::Entry::pool is COUNT (= 32),
and when each HttpRequest instance upload one file,
it will call IpAddrInfo::Start to retrive one entry from the IpAddrInfo::Entry::pool,
but it did not release the entry back to IpAddrInfo::Entry::pool when the uploading file operation was done or fail,
then another HttpRequest instance will stuck in its member function HttpRequest::Dns,
tring to retrive one available entry from the IpAddrInfo::Entry::pool,

```
void HttpRequest::Dns()
{
 for(int i = 0; i <= Nvl(GetTimeout(), INT_MAX); i++) {
  if(!addrinfo.InProgress()) {
   StartConnect();
   return;
  }
  Sleep(1);
 }
}
```

So, i made some modification, and the whole file is uploaded as attechment, hope that helps.

Modify from upp r7550

```
bool HttpRequest::Do()
{
 int c1, c2;
 switch(phase) {
 case BEGIN:
  retry_count = 0;
  redirect_count = 0;
  start_time = msecs();
  GlobalTimeout(timeout);
 case START:
  Start();
  break;
 case DNS:
  Dns();
  break;
 case SSLPROXYREQUEST:
  if(SendingData())
   break;
  StartPhase(SSLPROXYRESPONSE);
  break;
 case SSLPROXYRESPONSE:
  if(ReadingHeader())
   break;
  ProcessSSLProxyResponse();
  break;
 case SSLHANDSHAKE:
  if(SSLHandshake())
   break;
  StartRequest();
  break;
 case REQUEST:
  if(SendingData())
   break;
  StartPhase(HEADER);
  break;
 case HEADER:
  if(ReadingHeader())
   break;
  StartBody();
  break;
 case BODY:
  if(ReadingBody())
   break;
  Finish();
  break;
 case CHUNK_HEADER:
  ReadingChunkHeader();
```

```
 break;
case CHUNK_BODY:
 if(ReadingBody())
  break;
 c1 = TcpSocket::Get();
 c2 = TcpSocket::Get();
 if(c1 != '\r' || c2 != '\n')
  HttpError("missing ending CRLF in chunked transfer");
 StartPhase(CHUNK_HEADER);
 break;
case TRAILER:
 if(ReadingHeader())
  break;
 header.ParseAdd(data);
 Finish();
 break;
case FINISHED:
case FAILED:
 WhenDo();
 return false;
default:
 NEVER();
}

if(phase != FAILED) {
 if(IsSocketError() || IsError())
  phase = FAILED;
 else
 if(msecs(start_time) >= timeout) {
  HttpError("connection timed out");
  phase = FAILED;
 }
 else
 if(IsAbort()) {
  HttpError("connection was aborted");
  phase = FAILED;
 }
}

if(phase == FAILED) {
 if(retry_count++ < max_retries) {
  LLOGS("HTTP retry on error " << GetErrorDesc());
  start_time = msecs();
  GlobalTimeout(timeout);
  StartPhase(START);
 }
}
WhenDo();
```

```
 return phase != FINISHED && phase != FAILED;
}

to
bool HttpRequest::Do()
{
 int c1, c2;
 switch(phase) {
 case BEGIN:
  retry_count = 0;
  redirect_count = 0;
  start_time = msecs();
  GlobalTimeout(timeout);
 case START:
  Start();
  break;
 case DNS:
  Dns();
  break;
 case SSLPROXYREQUEST:
  if(SendingData())
   break;
  StartPhase(SSLPROXYRESPONSE);
  break;
 case SSLPROXYRESPONSE:
  if(ReadingHeader())
   break;
  ProcessSSLProxyResponse();
  break;
 case SSLHANDSHAKE:
  if(SSLHandshake())
   break;
  StartRequest();
  break;
 case REQUEST:
  if(SendingData())
   break;
  StartPhase(HEADER);
  break;
 case HEADER:
  if(ReadingHeader())
   break;
  StartBody();
  break;
 case BODY:
  if(ReadingBody())
   break;
  Finish();
```

```
 break;
case CHUNK_HEADER:
 ReadingChunkHeader();
 break;
case CHUNK_BODY:
 if(ReadingBody())
  break;
 c1 = TcpSocket::Get();
 c2 = TcpSocket::Get();
 if(c1 != '\r' || c2 != '\n')
  HttpError("missing ending CRLF in chunked transfer");
 StartPhase(CHUNK_HEADER);
 break;
case TRAILER:
 if(ReadingHeader())
  break;
 header.ParseAdd(data);
 Finish();
 break;
case FINISHED:
case FAILED:
 WhenDo();
 return false;
default:
 NEVER();
}

if(phase != FAILED) {
 if(IsSocketError() || IsError())
  phase = FAILED;
 else
 if(msecs(start_time) >= timeout) {
  HttpError("connection timed out");
  phase = FAILED;
 }
 else
 if(IsAbort()) {
  HttpError("connection was aborted");
  phase = FAILED;
 }
}

if(phase == FAILED) {
 if(retry_count++ < max_retries) {
  LLOGS("HTTP retry on error " << GetErrorDesc());
  start_time = msecs();
  GlobalTimeout(timeout);
  StartPhase(START);
```

```
  }
 }
 if( phase == FINISHED || phase == FAILED ) {
  addrinfo.Clear();
 }
 WhenDo();
 return phase != FINISHED && phase != FAILED;
}
```

File Attachments

1) Http.cpp, downloaded 493 times