Subject: cmdsrv does not work
Posted by cbpporter on Tue, 02 Jun 2009 19:25:50 GMT
View Forum Message <> Reply to Message

I needed a HTTP server so I thought I'll look over cmdsrv. But it does not work. It constructs the result correctly, but it will never get to the browser which will hang waiting for the response.

And after first request the server.wait call will no longer wait 1 second; I'm guessing because of the outstanding request, but I'm no expert on socket timeouts. (BTW, this is a strange way to do a server: doesn't it introduce up to a second lag? Or maybe it exits sooner if a request is gotten).

Subject: Re: cmdsrv does not work
Posted by rylek on Tue, 02 Jun 2009 21:07:05 GMT
View Forum Message <> Reply to Message

Hello!

I've just synced the SVN (in order to ensure that I've got up to date versions of all sources), rebuilt the cmdsrv and it seems to me to run without any difficulties, I see the generated page correctly both in Mozilla and Explorer. Please don't get offended, but isn't it by any chance possible that you were testing across a firewall which ate the server output? In any case the pending data should be flushed in the DelayedWrite method.

As concerns the 1 second Wait method call, this actually means 'wait at most 1 second but return immediately when a socket request arrives or a delayed write socket buffer gets freed to enable further data sending'. The 1 second timeout is just used to refresh the server status line in its command window and to check for server shutdown requests (Ctrl+C or 'kill' under Linux).

Regards

Tomas

Subject: Re: cmdsrv does not work
Posted by cbpporter on Tue, 02 Jun 2009 22:44:05 GMT
View Forum Message <> Reply to Message

Hi Tomas,

I believe I haven't had the pleasure of speaking with you directly before. Nice to meet you (virtually)!

I'm not offended but I did check my firewall settings.

Actually I found the problem after seeing you post. I was missing the delayed write in my sources.

I do always check latest SVN on problems, but unfortunately my "examples" nest was out-of-date, probably the one from 2008.1. After update it works fine now. Sorry, it was my mistake, but with so many different folders and parallel U++ installations it is easy to miss updating one of them, especially one that is that is so rarely used by me like the examples nest.

Since you are the author of this examples and possibly of HttpServer, I have a question: is this mechanism of semi-blocking wait and non blocking accept enough to handle heavy loads with longer response build times or should I create treads to handle each request?

Thank You,
Raul

Subject: Re: cmdsrv does not work
Posted by rylek on Thu, 04 Jun 2009 08:00:08 GMT
View Forum Message <> Reply to Message

Hello Raul!

Nice to meet you too, the pleasure is mine. I'm glad you've nailed the cause of the trouble, I know perfectly what you mean by "many folders and parallel U++ installations".

As concerns server throughput and possible parallelization, I simply don't know. I didn't write Google, if you know what I mean. The primary orientation at single-threaded solutions possibly with minor throughput enhancements like the delayed write cache was to a substantial extent caused by the fact that at the time of original coding of the HttpServer the multithreading support in U++ was still very rudimentary (politely speaking).

As you surely know by yourself, there are soooo many silly quirks to watch out for when coding in parallel, that [at least at an initial stage of a project] I would surely recommend to avoid multithreading unless absolutely necessary. Of course this depends on the kind of application you're writing. In case of heavy database transactions or waits on third-party web sources to complete your request, of course it would be silly to let a user wait for two minutes just to receive a two kilobyte welcome screen. (Not mentioning the browser would have timed out long before receiving the data.)

However there are also limits on the bandwidth of the outgoing network lines, even on the sockets themselves (as Mirek noted some time ago, if you refuse to suppress the recommended 30 second socket 'cooling' period after use, with 64K sockets available you are limited to about 2 requests per millisecond in any case), so it would be just as silly to spend a fortnight devising a super-smart super-parallel algorithm responding in a few microseconds, for instance.

Also please remember that web servers are tricky beasts by nature. In contrast with ordinary desktop applications which can often live even with certain more benign forms of leaks like missing internal cleanups, and whose crashes can be quite often well localized and simulated later on in a debugging environment, with a server running for a long period of time, perhaps at a remote hosting site, the situation is much more complicated. Very often all you can do is to rely on various log files and crash files, and retracing the exact sequence of events which led to the

failure is usually out of question.

In any case, from the point of view of U++, I believe the HttpServer object as such should be more or less adaptable to a multithreaded scheme with working threads handling the individual requests (represented by the HttpRequest object). If you decide to go this way, I'll be glad to hear from you concerning your experiences and perhaps improve the server object if it turns up that its current implementation is not usable in this way.

Regards

Tomas
