Subject: Re: New hash folding function...
Posted by mirek on Mon, 21 Jan 2008 18:07:57 GMT
View Forum Message <> Reply to Message

gprentice wrote on Mon, 21 January 2008 06:06Have you seen this article and tried the Jenkins One-at-a-time hash?
http://en.wikipedia.org/wiki/Hash_table

The wikipedia article also contains links for ways of testing a hash function.

http://en.wikipedia.org/wiki/Chi-square_test

Avalanche test code in C#
http://home.comcast.net/~bretm/hash/11.html

I'm not saying you should try these tests - just pointing them out

BTW - why does it take 2^n steps to map the hash code to the mapping space range = I thought you just masked the hash code with 2^something minus 1.

Well, in U++, things are a little bit more complicated.

Usually, in computer science, hashing means mapping the key value to target space. Anyway, that means recomputing all key values when target space changes -> slow.

Therefore U++ computes hashes just once, using modified FVN algorithm - modification is that instead of hashing individual bytes, it goes by dwords (if possible). Leads to slightly lower quality but much faster hash.

This is stored and then this 32bit quatity has to be mapped to the real target space. This secondary mapping is the subject of my recent efforts

Anyway, thanks for links, I will try to apply some testing mentioned to add to my testsuite....

As about 2^n, it is not 2^n, but simpy >16 CPU cycles to perform modulo operation of 32-bit value, because current CPUs compute 2 bits of result per cycle. Which is now quite a lot, compared to other oprations involved in Index.

BTW, considering all this, you have always to watch correct tradeoffs. Many hash functions have much better quality than what we use now. However, for use in hash-table, what is the point to have perfect hashing function, when it takes much longer time than equality comparison?

Quote:
What's wrong with the URL mechanism ?!?

What is URL mechanism?

Mirek