Subject: Which parts of Esc are the biggest reasons of its slowliness?
Posted by fudadmin on Tue, 15 Aug 2006 08:29:51 GMT
View Forum Message <> Reply to Message

Which parts of Esc and/or CParser are the biggest reasons of its slowliness?
I've started re-writing some parts of my favourite interpreter (and its U++ port...). To remove some limitations and make the executable smaller I want to use as much as possible code from U++.
Then, maybe I could offer some speed improvements to Esc, too?
My suspects or parts of interest:
StringStream:
1. too many function calls get(c) when e.g get32? (actually I found the analog for me - get32be)
2. some "inline" are ignored by the compiler (I've read that you would need "force inline" for MS compilers...)
3. because the raw data are not contigeous in memory but with too many links (or something...)

GLOBAL macro...

in CParser:
C syntax {}
e.g, I guess, using If ... endif for ...endfor could speed the things up? (I will be using this anyway and in compiled scripts just 1 unsigned char.)

Anything else to consider?
Pointers vs references? Type casting?
And, Mirek, (or anyone else), do you have your suspects in an approximate % order?.

P.S.
What is better, when and why (I'm confused because of UPP_HEAP)?

```
 U8 m_CodeBuffer[4];
  m_CodeBuffer[0]=s.Get8();
  m_CodeBuffer[1]=s.Get8();
  m_CodeBuffer[2]=s.Get8();
  m_CodeBuffer[3]=s.Get8();

 int m_Index=0;
 result_int32be =(((U8)(m_CodeBuffer[m_Index])     << 24) |
   ((U8)(m_CodeBuffer[m_Index + 1]) << 16) |
   ((U8)(m_CodeBuffer[m_Index + 2]) <<  8) |
   (U8) m_CodeBuffer[m_Index + 3]);
```

or

```
int get32int()
{
```

```
 U8* m_CodeBuffer = new U8[4];
 int m_Index=0;
 int x =(((U8)(m_CodeBuffer[m_Index])     << 24) |
   ((U8)(m_CodeBuffer[m_Index + 1]) << 16) |
   ((U8)(m_CodeBuffer[m_Index + 2]) <<  8) |
   (U8) m_CodeBuffer[m_Index + 3]);
 delete [] m_CodeBuffer;
 return x;
}
```

does operator *new* changes its behaviuor in case of USE_UPP_HEAP? What are pluses/minuses of USE_MALLOC in relation if I use malloc - realloc in my code? (I know not to mix *new* and free()...)

Are there any docs about memory things in upp?

P.S.2 Or, Mirek, what about sharing some of your favourite links with our community ?
Thanks in advance.

Subject: Re: Which parts of Esc are the biggest reasons of its slowliness?
Posted by mirek on Tue, 15 Aug 2006 10:25:41 GMT
View Forum Message <> Reply to Message

Actually, the main reasons why Esc is relatively slow IMO are:

- data model. While it is very simple and very effective, it is also quite slow (esp. the way how strings are stored).

- fact that it is interpreted based on source text only. on-the-fly compilation to intermediate language would make it faster, but would add thousands of lines.

Note that there are no reasons why Esc is slow in the Core (well, I have plans how to speed up String implementation, but there is nothing wrong with current one). In fact, Core almost never makes performance tradeoffs.

Mirek

Subject: Re: Which parts of Esc are the biggest reasons of its slowliness?
Posted by mirek on Tue, 15 Aug 2006 10:34:38 GMT
View Forum Message <> Reply to Message

About U++ heap: Memory allocator of U++ is I believe the most optimal possible algorithm&implementation (inspired by Boehm's GC). In fact, if I would have time, it would be

worth to publish paper just about techniques used there

Just some highlights:

- Small-block fast allocation+dealocation path (used in majority of cases) has about 20+20 assembler instruction (plus synchronization in MT).

- There is less than one byte of management data overhead per small block. That also means that the smallest block size can be 4 bytes without problems (on 32-bit platform, on 64 it is 8 bytes). Actually, smaller blocks have lower overhead than larger ones, unlike classic allocators.

- Small-block fragmentation for real-world cases is limited by absolute value (I am not sure at the moment, but if I remember last tests well, averge maximum fragmentation limit is about 100KB).

Now USE_MALLOC is development macro that turns this high-efficient U++ heap off and uses regular malloc instead...

Mirek

Subject: Re: Which parts of Esc are the biggest reasons of its slowliness?
Posted by fudadmin on Tue, 15 Aug 2006 10:58:08 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 15 August 2006 11:25Actually, the main reasons why Esc is relatively slow IMO are:

1. - data model. While it is very simple and very effective, it is also quite slow (esp. the way how strings are stored).

2. - fact that it is interpreted based on source text only. on-the-fly compilation to intermediate language would make it faster, but would add thousands of lines.

3. Note that there are no reasons why Esc is slow in the Core (well, I have plans how to speed up String implementation, but there is nothing wrong with current one). In fact, Core almost never makes performance tradeoffs.

Mirek

1 and 3.
Your words about Core String things makes me more relaxed...
It looks now to me that I formed my prejudice about Strings from the Esc model!

1. and 2. Anyway, I'm experimenting with them... I'll ask later, then.
Thanks.
