

code efficiency

Introduction

This presentation talks about the impact of a “small” difference on the efficiency of a web application and on the load created on the server.

While this presentation is based on PHP, the results are similar for MySQL queries or JavaScript scripts.

the subject

The subject of this presentation is the use of the == operator on strings versus integers and how this impacts the load of the server's CPU.

Below, you will find the two cases that we will discuss:

```
<?php
$debug = "0";

if ( $debug == "0" ) {
    exit( -1 );
}
```

```
<?php
$debug = 0;

if ( $debug == 0 ) {
    exit( -1 );
}
```

behind the scene

PHP is written in C/C++. Everything we write in PHP translates into a piece of code written in C. Without actually going into the PHP source code we can make an assumption about how the above code would be implemented, excluding a lot of the tests and overhead induced by the fact that PHP is a scripting language.

Here's how it would look like in C:

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *debug = "0";

    if ( strcmp( debug , "0" ) == 0 ) {
        return -1;
    }

    return 0;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int debug = 0;

    if ( debug == 0 ) {
        return -1;
    }

    return 0;
}
```

Right from the start we notice that in the string comparison case we have a function call and in the integer case it is almost similar to the PHP version.

how does that translate

Of course, the CPU does not understand C so we'll take a pick at what Microsoft's C Compiler outputs for our code sequences:

string

```
char *debug = "0";
004113BE  mov     dword ptr [debug],offset string "0"

if ( strcmp( debug , "0" ) == 0 ) {
004113C5  push   offset string "0" (41573Ch)
004113CA  mov     eax,dword ptr [debug]
004113CD  push   eax
004113CE  call   @ILT+165(_strcmp) (4110AAh)
004113D3  add     esp,8
004113D6  test   eax,eax
004113D8  jne    wmain+3Fh (4113DFh)
    return -1;
004113DA  or     eax,0FFFFFFFFh
004113DD  jmp    wmain+41h (4113E1h)
}

return 0;
004113DF  xor     eax,eax
}
004113E1  pop     edi
```

integer

```
int debug = 0;
0041138E  mov     dword ptr [debug],0

if ( debug == 0 ) {
00411395  cmp     dword ptr [debug],0
00411399  jne    wmain+30h (4113A0h)
    return -1;
0041139E  or     eax,0FFFFFFFFh
0041139E  jmp    wmain+32h (4113A2h)
}

return 0;
004113A0  xor     eax,eax
}
004113A2  pop     edi
```

strcmp – the algorithm

Algorithm:

```
int strcmp ( src , dst )
    unsigned char *src;
    unsigned char *dst;
{
    int ret = 0 ;

    while( ! (ret = *src - *dst) && *dst)
        ++src, ++dst;

    if ( ret < 0 )
        ret = -1 ;
    else if ( ret > 0 )
        ret = 1 ;

    return( ret );
}
```

The algorithm which implements strcmp is outlined on the left.

Basically the strings are walked with two pointers and each element is compared with its counterpart to find out if the strings match and if not, how they differ.

strcmp - the implementation

```
strcmp:                                     donene:
1026F7D0  mov     edx,dword ptr [esp+4]             1026F814  sbb     eax,eax
1026F7D4  mov     ecx,dword ptr [esp+8]             1026F816  shl     eax,1
1026F7D8  test   edx,3                             1026F818  add     eax,1
1026F7DE  jne    dopartial (1026F81Ch)             1026F81B  ret
dodwords:                                   dopartial:
1026F7E0  mov     eax,dword ptr [edx]              1026F81C  test   edx,1
1026F7E2  cmp    al,byte ptr [ecx]                 1026F822  je     dword (1026F83Ch)
1026F7E4  jne    donene (1026F814h)                1026F824  mov    al,byte ptr [edx]
1026F7E6  or     al,al                              1026F826  add    edx,1
1026F7E8  je     doneeq (1026F810h)                 1026F829  cmp    al,byte ptr [ecx]
1026F7EA  cmp    ah,byte ptr [ecx+1]                1026F82B  jne    donene (1026F814h)
1026F7ED  jne    donene (1026F814h)                1026F82D  add    ecx,1
1026F7EF  or     ah,ah                              1026F830  or     al,al
1026F7F1  je     doneeq (1026F810h)                 1026F832  je     doneeq (1026F810h)
1026F7F3  shr    eax,10h                            1026F834  test   edx,2
1026F7F6  cmp    al,byte ptr [ecx+2]                1026F83A  je     dodwords (1026F7E0h)
1026F7F9  jne    donene (1026F814h)                dword:
1026F7FB  or     al,al                              1026F83C  mov    ax,word ptr [edx]
1026F7FD  je     doneeq (1026F810h)                 1026F83F  add    edx,2
1026F7FF  cmp    ah,byte ptr [ecx+3]                1026F842  cmp    al,byte ptr [ecx]
1026F802  jne    donene (1026F814h)                 1026F844  jne    donene (1026F814h)
1026F804  add    ecx,4                              1026F846  or     al,al
1026F807  add    edx,4                              1026F848  je     doneeq (1026F810h)
1026F80A  or     ah,ah                              1026F84A  cmp    ah,byte ptr [ecx+1]
1026F80C  jne    dodwords (1026F7E0h)               1026F84D  jne    donene (1026F814h)
1026F80E  mov    edi,edi                             1026F84F  or     ah,ah
doneeq:                                       1026F851  je     doneeq (1026F810h)
1026F810  xor    eax,eax                             1026F853  add    ecx,2
1026F812  ret                                         1026F856  jmp    dodwords (1026F7E0h)
1026F813  nop
--- No source file -----
```

what next ?

Each asm instruction translates into a series of micro instructions on the CPU.

This means that for everytime the \$debug variable is checked troughout an application, an entire mecanism fires up its engines to solve what could be an extremly easy task.

In the case of the INTEGER comparision, a TOTAL ammount of 4 asm instructions are executed.

In the case of the STRING comparision, an ammount of 9 asm instructions are executed only in the main block, which includes a function call. The strcmp function has 55 asm instructions which are executed in various scenarios and which include loops. On an average testing 1 digit will execute $9 + 17 = 26$ asm instructions of various complexity levels.

conclusion

For an average of 10 such checks on a webpage this results in:

INTEGER:
40 asm instr

STRING:
260 asm instr

DIFFERENCE:
220 asm instructions

areas of applicability

While this study was made using PHP, the same concepts apply to:

- SQL queries where string data types are used for comparison, order, grouping etc.
- JavaScript snippets
- Any programming language
- Even Excel formulas!
- Everything related to programming :-)