Native types in Perl 6 - a quick look

David Warring March 2017

Native variables, Blobs/Bufs and NativeCall

About me (career)

- Web/Learning Management Systems/Print Production
- C programmer, 90s
- Perl last 10 years
- Dunedin resident, for three years
- Work remotely

Perl 6 - Project

- Developed from RFCs / community
- Complete redesign and redevelopment
- Multi everything (method, language, paradigm, threaded, VM)
- Grammars, Unicode
- Specified/Engineered/Tested
- Gradual uptake

About me (Perl 6)

- 4 years part-time
- Distributed team IRC/email
- Works off Rakudo blead
- Tests (Roast)
- Tickets
- Low level PDF
- Looking at Early Adoption

Perl 6 Gradual Typing

Untyped Perl 6 scalars are similar to Perl 5. They can hold just about anything:

```
my $v = 37 + 5;
$v = "hi";
```

But in Perl 6 we can declare types:

```
my Int $v = 37 + 5;
try $v = "hi"; # Type check failed in assignment to $v...
```

We can also declare type on Arrays, Hashes (and other containers). These are applied to individual elements:

```
my UInt @a = 3, 5, 7, 9; # array of positive integers
try @a.push: -99; # Type check failed in assignment ...
```

Perl 6 Native Types

Native types are built into Perl 6.

Declared with lowercase names:

```
my uint8 $i = 248; # unsigned byte
$i += 42;
say $i; # 34
```

`int8`, `int16`, `num` (float), `str` (char*), ...etc

Also pointers.

Native arrays. `my uint8 @state = 0..255;`

Native arrays are compact and contain just one data-type. Similar to arrays in languages such as C#, Java or C.

Buf's and Blobs

These are typically used to read and write binary data.

- They may be used to create and save Unicode strings in different encodings (utf-8, utf-16, etc).
- For our purposes they are commonly used to pass data to native functions.

```
my buf8 $in = '/dev/urandom'.IO.open.read(512, :bin);
my buf8 $out .= allocate(31);
do-stuff($in, $in.bytes, $out);
'/dev/null'.IO.open(:w).write($out);
```

NativeCall

Just a special subroutine or method definition. Associated with a shared library.

Example:

Adding MD5 Digest to OpenSSL module.

Consider `MD5` digest definition from `openssl/md5.h`:

```
#define MD5_DIGEST_LENGTH 16
# ...
unsigned char *MD5(const unsigned char *d, size_t n,
                   unsigned char *md);
```

Perl 6

```
use OpenSSL::NativeLib;
use NativeCall;

unit module OpenSSL::Digest;

our constant MD5_DIGEST_LENGTH    = 16;

sub MD5( Blob, size_t, Blob )
    is native(&gen-lib)    { ... }

sub md5(Blob $msg) is export {
     my $digest = buf8.allocate(MD5_DIGEST_LENGTH);
     MD5($msg, $msg.bytes, $digest);
     $digest;
}

# also sha1, sha256
```

Testing this module:

```
use Test;
use OpenSSL::Digest;

my Blob $msg = 'abc'.encode: 'latin-1';
my Str $digest = md5($msg)>>.fmt("%02x").join;
is $digest, '900150983cd24fb0d6963f7d28e17f72', "md5";

done-testing;
```

At least 10x+ speedup (Rakudo 2017.03) from using a native function over pure Perl.

C Code [buf.c]:

```
extern void buf_unpack_16(uint8_t *in, uint16_t *out, siz
  size_t i;
  size_t j = 0;
  for (i = 0; i < in_len; j++) {
    out[j]  = in[i++] << 8;
    out[j] += in[i++];
  }
}
```

Perl6 benchmark:

```
    use Lib::Buf;
    # -*- setup -*-
    my $buf = buf8.new: (^255).pick xx 100_000;
    warn;
    my uint $n = $buf.bytes div 2;
    # -*- timed -*-
    my $t = now;
    my buf16 $nw;
    for 1..300 {
    warn;
       $nw = Lib::Buf.unpack($buf, 16);
```

Perl 6 code migrated from C.

Not fast yet (Rakudo 2017.03).

JIT mostly missing (Just in Time Compilation)

```
    sub unpack($buf, @nw) {
        my uint $i = 0;
        my uint $j = 0;
        my uint $n = +@nw;
        while $j < $n {
            @nw[$j++] = $buf[$i++] * 256 + $buf[$i++];
        }
    };

    my $buf = buf8.new: (^255).pick xx 100_000;
    my uint $n = $buf.bytes div 2;
    my uint16 @nw[$n];
    # -*- timed -*-
    my $t = now;
    unpack($buf, @nw);
    say now - $t; # ~ 1sec
```

- Keeping it as Pure Perl

Conclusions

Native Types

- Easy integration via NativeCall
- Recommended when performance needed
- Modest performance improvements (2017.03)
- But should be much faster
- Can code natively, if needed
- Perl 6 / C work well together
    - prototyping / harnessing / test-suite

Further Discussion:

- Much more to NativeCall
  (structs, unions, allocation, callbacks,..)
- Other native languages