Perl6

The Joy of NativeCall

Presented to http://oc.pm.org/ on 2016-02-29
Erick Jordan (erickfjordan@gmail.com)

Perl5 External Subroutine “XS”

In Perl5 the way you interface with native code (C libraries) is with XS. XS is a language which allows you to describe the mapping of how a C function is used and how a corresponding Perl routine is used. The output of XS is a dynamic (or static) library which forms the bridge between Perl and the native library. Writing modules in XS takes effort. There are other tools such as SWIG which have their own XS-like language which make building libraries easier, but still there are drawbacks.

Perl5 External Subroutine “XS”

Drawbacks such as …

- Carting your bridge library to wherever your Perl installation resides.
- Binary compatibility with Perl installation (32 bit vs 64bit vs Win32/64 …).
- New releases of Perl may break binary compatibility with your bridge library.

The Joy of NativeCall

Perl6 puts a lot of effort into letting you be lazy when it comes to interfacing with native libraries. The NativeCall module builds on a set of native calling primatives in NQP (Not Quite Perl), adding mapping of Perl 6 signatures and various other traits to make working with native libraries an easier experience.

When reading about NativeCall you may see references to a project name **Zavolaj!** This is a Slovak word which translates to the imperative “call!”

Starting in 2015.02, NativeCall ships with the Perl6 compiler so there’s no need to install it separately.

A Simple Example

```
use NativeCall;
sub fork() returns uint32 is native { ... }

my $children = 15;
for 1 .. $children -> $child {
    my $pid = fork();
    if $pid {
        print "created child $child process $pid. ";
        sleep 1; print "snore. ";
    }
    else {
        for $child .. $children { sleep 1; print "yawn $child. "; }
        exit 0;
    }
}
```

A Simple Example

Here the C function fork() is already loaded as part of the standard library. All that's needed is a declaration of sub fork() in your code:

```
sub fork() returns uint32 is native { ... }
```

The declaration looks like a regular perl6 routine but with the added "native" trait which says the sub is actually defined in a native library. To learn how to declare your sub you typically look up its signature in the header files that usually ship with the native library. Using the online man pages is a great starting point e.g. "man fork".

Loading a Native Library

Under the hood, NativeCall uses dlopen() to dynamically load an external library. This is accomplished with the “native” trait in a sub declaration. So for example …

```
use NativeCall;
sub some_argless_function() is native('something') { * }
```

The first time you call `some_argless_function` “libsomething.so” is loaded by dlopen(). Subsequent calls will be faster since the symbol handle to “libsomething.so” is retained.

Argument Passing

Normal Perl6 signatures and the “returns” trait are used to convey the type of arguments a native function expects and what it returns.

```
sub add(int32, int32) returns int32 is native(‘calculator’) { … }
```

Here are some of the supported types which may be used (likely to grow over time) …

```
int8                (char in C)
int16               (short in C)
int32               (int in C)
int64               (64-bit explicit int in C like int64_t in C99)
long                (32- or 64-bit, depends what long means locally)
```

Argument Passing

```
Longlong              (at least 64bits, what long long means locally)
num32                 (float in C)
num64                 (double in C)
Str                   (C string)
CArray[int32]         (int* in C, an array of integers)
Pointer[void]         (void* in C, can point to all other types)
bool                  (bool from C99)
size_t                (size_t in C)
```

WARNING: Do not mix these data types with Perl6 data types. Use them in your native sub declarations and C Struct (“repr” class) definitions. For example, don’t do this .. `sub add(Int, Int) returns Int is native('calculator') { … }`

The “repr” Trait

Using trait “repr” it is possible to declare a normal looking Perl6 class that, under the hood, stores its attributes in the same way a C compiler would lay them out in a struct definition. Example from <pwd.h>

```
struct passwd {
  char *pw_name;   /* username */
  char *pw_passwd; /* user password */
  uid_t pw_uid;    /* user ID */
  gid_t pw_gid;    /* group ID */
  char *pw_gecos;  /* user information */
  char *pw_dir;    /* home directory */
  char *pw_shell;  /* shell program */
};
```

The “repr” Trait

Here is how you represent C struct passwd as a Perl6 class …

```
my class PwStruct is repr('CStruct') {
    has Str $.pw_name;
    has Str $.pw_passwd;
    has uint32 $.pw_uid;
    has uint32 $.pw_gid;
    has Str $.pw_gecos;
    has Str $.pw_dir;
    has Str $.pw_shell;
};
sub getuid() returns uint32   is native {...};
sub getpwuid(int32 $uid) returns PwStruct is native {...};

my $pw = getpwuid(getuid());
say "name   : " ~ $pw.pw_name;
say "passwd : " ~ $pw.pw_passwd;
```

The “rw” Trait

When a signature of your native function needs a pointer to some native type (int, uint, etc.) use the “rw” trait in your argument.

```
# C prototype is char *ctime(const time_t *timep);
sub ctime(uint32 is rw) returns Str is native { ... }

# C prototype is time_t time();
sub time() returns uint32 is native { ... }

# C prototype is struct tm *localtime(const time_t *timep);
sub localtime(uint32 is rw) returns tm is native { ... }

my uint32 $timep = time();
say chomp(ctime($timep));

my $tm = localtime($timep);
say $tm;
```

Miscellany

A perl script called win32-api-call.p6 shows an example Windows API call
See http://github.com/jnthn/zavolaj/tree/master/examples

NativeCall offers support to use class and methods from C++.
See https://github.com/rakudo/rakudo/blob/nom/t/04-nativecall/13-cpp-mangling.t

NativeCall offers type casting of certain native types using nativecast(). You can open up Pandora's box with nativecast and do things you're not suppose to (i.e. increment pointers)

Perl6 Modules Using NativeCall

https://modules.perl6.org

Name	Description
DBIish	Database connectivity for Perl6
Compress::Brotli	Brotli compression using NativeCall
Crypt::Bcrypt	An implementation of bcrypt (Blowfish)
FastCGI::NativeCall	An implementation of FastCGI using NativeCall
GTK::Simple	Simple GTK 3 binding using NativeCall
OpenCV	OpenCV Bindings using NativeCall

References

https://doc.perl6.org/language/nativecall

https://perl6advent.wordpress.com/2010/12/15/day-15-calling-native-libraries-from-perl-6/

https://github.com/jnthn/zavolaj

https://modules.perl6.org

Thank You.