Using NIFs for Better SHA-2 Functions

Steve Vinoski

London Erlang User Group Meeting

3 Mar 2011

Native Implemented Functions (NIFs)

- Functions that appear as regular Erlang functions but are implemented in C/C++
- Officially supported from version R14B
- Typically used for
  - accessing libraries or platform features not available through Erlang
  - performance

SHA-2 in Erlang

- SHA-{224,256,384,512} cryptographic hash functions
  - http://en.wikipedia.org/wiki/SHA-2
- Not available in Erlang standard library
- I first implemented them in pure Erlang a few years ago, for fun
- Functional and correct, but slow

SHA-2 With NIFs

- Recently reimplemented my original SHA-2 Erlang code using NIFs
  - changed function names to match those in Erlang crypto module
  - added init/update/final variants

Usage

```
Data = <<“binary or iolist”>>,
Digest1 = erlsha2:sha256(Data),

%% or

Ctx = erlsha2:sha256_init(),
NCtx1 = erlsha2:sha256_update(Ctx, Data1),
NCtx2 = erlsha2:sha256_update(NCtx1, Data2),
%% can do more updates here
Digest2 = erlsha2:sha256_final(NCtx2),
```

NIF Loading

```
-module(erlsha2).
-export([sha224/1, sha256/1, sha384/1, sha512/1]).
-export([sha224_init/0, sha224_update/2, sha224_final/1]).
-export([sha256_init/0, sha256_update/2, sha256_final/1]).
-export([sha384_init/0, sha384_update/2, sha384_final/1]).
-export([sha512_init/0, sha512_update/2, sha512_final/1]).
-version(2.0).

-on_load(init/0).
```

init/0

```
init() ->
    SoName = filename:join(case code:priv_dir(?MODULE) of
                               {error, bad_name} ->
                                   %% this is here for testing purposes
                                   filename:join(
                                     [filename:dirname(
                                        code:which(?MODULE)),"..","priv"]);
                               Dir ->
                                   Dir
                           end, atom_to_list(?MODULE) ++ "_nif"),
    erlang:load_nif(SoName, 0),
    ok.
```

- Our init/0 returns ok so we use Erlang funs if NIF load fails, normally just return load_nif

NIF Functions

- Any exported functions provided by the loaded NIF library replace those in the Erlang module
- Any exported functions not provided by the NIF lib are expected to be provided by the Erlang module
- Allows mix-and-match of Erlang funs and native funs

Function Descriptors

```
static ErlNifFunc funcs[] = {
    {"sha224", 1, sha224},
    {"sha224_init", 0, sha224_init},
    {"sha224_update", 2, sha224_update},
    {"sha224_final", 1, sha224_final},

    /* same again for 256, 384, 512 */

};
```

NIF Init

```
ERL_NIF_INIT(
  erlsha2,              /* module name */
  funcs,                /* function descriptors */
  nifload,              /* load function */
  NULL,                 /* reload function */
  NULL,                 /* upgrade function */
  NULL)                 /* unload function */
```

Resource Objects

- These are NIF-specific native objects that can be passed back and from to/from Erlang
- They appear to Erlang as empty binaries
- Handy for storing state while making the caller responsible for it
- Not all NIFs will use resource objects

SHA-2 Resource

```
typedef struct {
    uint64_t bitlen;
    unsigned char bytes[2*PADDED_SIZE_5XX_BYTES];
    ErlNifBinary digest;
    size_t count;
    size_t size;
} Context;
```

SHA-2 Resource

```
typedef struct {
    uint64_t bitlen;
    unsigned char bytes[2*PADDED_SIZE_5XX_BYTES];
    ErlNifBinary digest;
    size_t count;
    size_t size;
} Context;
```

Stores the eventual hash result

Resource Destructor

```
static void
context_dtor(ErlNifEnv* env, void* obj)
{
    Context* ctx = (Context*)obj;
    if (ctx != NULL && ctx->digest.size > 0) {
        enif_release_binary(&ctx->digest);
    }
}
```

Resource Type

```
static int
nifload(ErlNifEnv* env, void** priv_data, ERL_NIF_TERM load_info)
{
    *priv_data = enif_open_resource_type(
                            env, NULL,"erlsha2_context",
                            context_dtor, ERL_NIF_RT_CREATE, NULL);
    return 0;
}
```

Resource Management

- Allocate resource objects via enif_alloc_resource
- Convert to Erlang term via enif_make_resource
- Release resource via enif_release_resource
- Add ref to resource via enif_keep_resource
- Each Alloc/Keep must have corresponding Release

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
           const ERL_NIF_TERM argv[])
{
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
Context* ctx = (Context*)enif_alloc_resource(
                                    ctx_type, sizeof(Context));
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
    Context* ctx = (Context*)enif_alloc_resource(
                                    ctx_type, sizeof(Context));
    context_init(ctx, H224, sizeof H224,
                        PADDED_SIZE_2XX);
    result = enif_make_resource(env, ctx);
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
    Context* ctx = (Context*)enif_alloc_resource(
                                    ctx_type, sizeof(Context));
    context_init(ctx, H224, sizeof H224,
                        PADDED_SIZE_2XX);
    result = enif_make_resource(env, ctx);
    enif_release_resource(ctx);
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
    Context* ctx = (Context*)enif_alloc_resource(
                                        ctx_type, sizeof(Context));
    context_init(ctx, H224, sizeof H224,
                        PADDED_SIZE_2XX);
    result = enif_make_resource(env, ctx);
    enif_release_resource(ctx);
    return result;
}
```

sha*_init Function

```
static ERL_NIF_TERM
hd224_init(ErlNifEnv* env, int argc,
                const ERL_NIF_TERM argv[])
{
    ERL_NIF_TERM result;
    ErlNifResourceType* ctx_type = (ErlNifResourceType*)
        enif_priv_data(env);
    Context* ctx = (Context*)enif_alloc_resource(
                                        ctx_type, sizeof(Context));
    context_init(ctx, H224, sizeof H224,
                        PADDED_SIZE_2XX);
    result = enif_make_resource(env, ctx);
    enif_release_resource(ctx);
    return result;
}
```

Handling Arguments

- Args are passed via argc/argv model
- Each arg is a const ERL_NIF_TERM
- Convert to native type via enif_get_* functions or enif_inspect_* functions
- Test types via enif_is_* functions

Handling Arguments

```
static ERL_NIF_TERM
XXX_final(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
```

Handling Arguments

```
static ERL_NIF_TERM
XXX_final(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    ContextUnion ctxu;
    ErlNifResourceType* ctx_type =
        (ErlNifResourceType*)enif_priv_data(env);
```

Handling Arguments

```
static ERL_NIF_TERM
XXX_final(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    ContextUnion ctxu;
    ErlNifResourceType* ctx_type =
        (ErlNifResourceType*)enif_priv_data(env);
    if (!enif_get_resource(env, argv[0], ctx_type, &ctxu.v))
        return enif_make_badarg(env);
```

Handling Arguments

```
static ERL_NIF_TERM
XXX_final(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    ContextUnion ctxu;
    ErlNifResourceType* ctx_type =
        (ErlNifResourceType*)enif_priv_data(env);
    if (!enif_get_resource(env, argv[0], ctx_type, &ctxu.v))
        return enif_make_badarg(env);
```

Badarg Caveat

- If you call enif_make_badarg, you MUST return its return value from your NIF function
- Not really documented in R14B01 but docs patched for R14B02
- Also added a new enif_is_exception() function, might be included in R14B02
- Other exception types might appear in the future

Handling Arguments

```
static ERL_NIF_TERM
XXX_final(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    ContextUnion ctxu;
    ErlNifResourceType* ctx_type =
        (ErlNifResourceType*)enif_priv_data(env);
    if (!enif_get_resource(env, argv[0], ctx_type, &ctxu.v))
        return enif_make_badarg(env);
    return context_fini(env, ctxu.c, DIGEST_SIZE_XXX,
                        shaXXX_chunk);
}
```

Performance

- Running erlsha2 test suite
  - without NIFs: 28 seconds
  - with NIFs: 0.75 second

Use With Care

- NIF crashes will take down the whole VM
- Slow NIFs will block the VM

For More Details

- http://www.erlang.org/doc/man/erl_nif.html
- https://github.com/vinoski/erlsha2