The Evolution of Erlang Drivers and the Erlang Driver Toolkit

Scott Lystig Fritchie
Snookles Music Consulting

October 7, 2002

How Do I Use This C Library with Erlang?

- Interface with: Dialogic IVR boards, Oracle, Sybase, MPI, Berkeley DB, mmap(), System V shared memory, ...

- Your options
  - Write the driver yourself: steep learning curve.
  - Re-implement the library in Erlang: steep effort curve.
  - Don't use Erlang

- There must be a better way ....

Language Extensibility: Tcl, Python, Perl

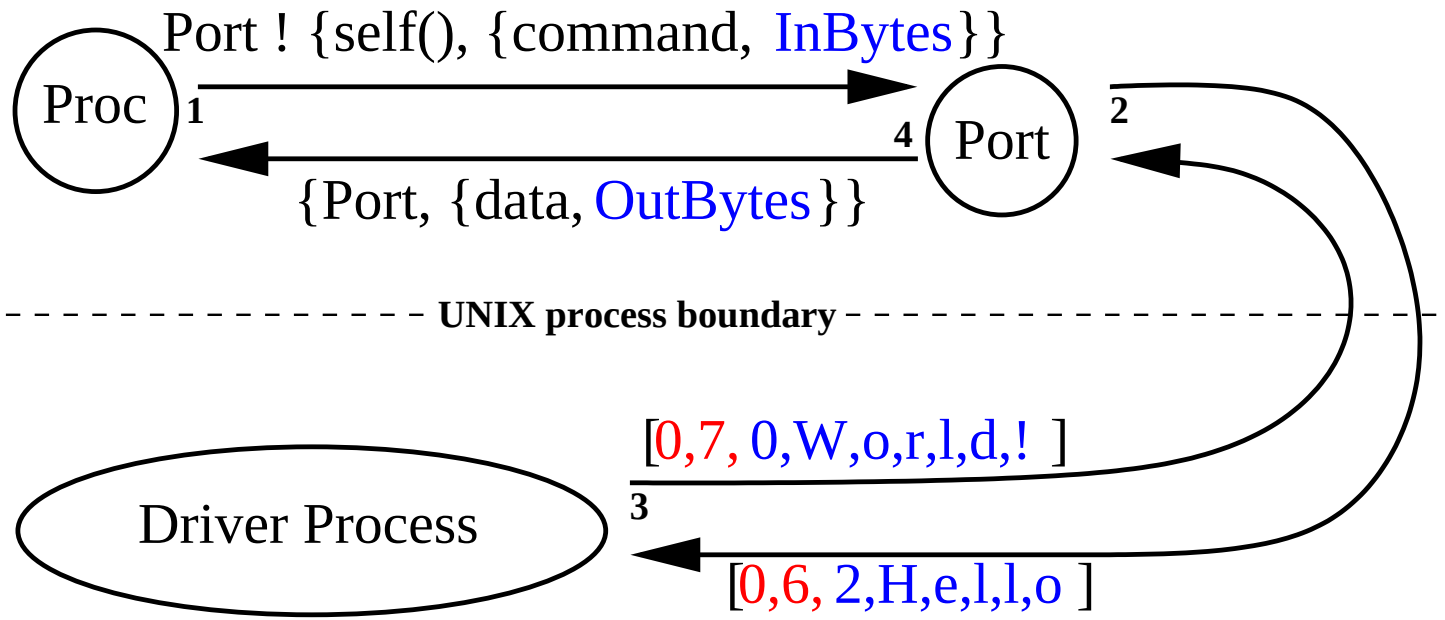
- All three have well-documented extensibility mechanisms:
  - "Callback registration" to add new command/function
  - Argument passing convention
  - Result return convention
  - Error conventions

```
#include "tcl.h"

static int
SLF_eq(ClientData ignored, Tcl_Interp *interp,
       int argc, char *argv[])
{
  char *result;

  if (argc != 3)
    return TCL_ERROR;
  result = !strcmp(argv[1], argv[2]) ? "Y" : "N";
  Tcl_SetResult(interp, result, TCL_STATIC);
  return TCL_OK;
}

int
Pkga_Init(Tcl_Interp *interp)
{
  Tcl_CreateCommand(interp, "slf_eq", SLF_eq,
    (ClientData) 0, (Tcl_CmdDeleteProc *) NULL);
  return TCL_OK;
}
```

Extending Erlang: The Port

- Avoid traumatizing Erlang programmers by making non-Erlang stuff look like Erlang.
  - Message passing to/from a port is the same as to/from a process.
- Driver evolution includes:
  - "Linked-in" driver executes in VM's address space: speed!
  - Ports return binary type: much more efficient!
  - Linked-in drivers return arbitrary terms: no serialization! (Only in one direction)

Simple Pipe Driver Diagram



Other Extension Techniques

- Do it yourself: Plain old network transport, serial ports, local file system, ...
- IG ... most EDTK-like, C-like specification file
- SWIG ... huge user base
- Erl_interface and "C nodes" ... Message passing "Looks just like Erlang"
- Sun Microsystem's RPC, OMG's CORBA, Microsoft's DCOM, and others ...

```
def filecopy(source, target)
  f1 = fopen(source, "r")
  f2 = fopen(target, "w")
  buffer = malloc(8192)
  nbytes = fread(buffer, 8192, 1, f1)
  while (nbytes > 0):
    fwrite(buffer, nbytes, 1, f2)
    nbytes = fread(buffer, 8192, 1, f1)
  free(buffer)
  fclose(f1)
  fclose(f2)
```

```
%module fileio
FILE *fopen(char *, char *);
int fclose(FILE *);
unsigned fread(void *, unsigned,
               unsigned, FILE *);
unsigned fwrite(void *, unsigned,
                unsigned, FILE *);
void malloc(int nbytes);
void free(void *ptr);
```

Constraints Imposed By Erlang

- Single-assignment semantics
- Data serialization through the port
  - Linked-in drivers can bypass this, but only in one direction
- Fault tolerance: resource leaks are eventually fatal
- Thread management
  - Asynchronous driver mechanism introduced in R7.

EDTK Design Goals

- Quickly develop a prototype
- Create a time-saver, not a labor-eliminator
- Support all driver types:
  - Pipe and linked-in drivers
  - Linked-in: synchronous and asynchronous execution of extension functions

GSLgen example

```
<?xml version="1.0"?>
<tree>
  <entity name="World">
    <greeting type="Hello"/>
  <entity/>
  <entity name="Planet">
    <greeting type="Salutations"/>
  <entity/>
</tree>
```

XML input file

```
.ignorecase = 0
#!/bin/sh

.for entity
.  for greeting
echo "$(.type), $(entity.name)!"
.  endfor
.endfor
```

GSLgen input schema file

```
#!/bin/sh

echo "Hello, World!"
echo "Salutations, Planet!"
```

GSLgen output

Step-By-Step Example: Calling fwrite(3)

Erlang:

```
example_drv:fwrite(["Hello, ", <<"world!">>], FilePtr).
```

XML:

```
<func name="fwrite">
  <arg name="ptr" ctype="char *" ser_type="binary" binlen2stash="0"/>
  <arg name="size" ctype="size_t" noerlcall="1"/>
  <arg name="nmemb" ctype="size_t" noerlcall="1" usestash="0"/>
  <arg name="filep" ctype="FILE *" ser_type="integer"/>
  <return ctype="size_t" name="ret_size_t" etype="integer"/>
  <hack place="post-deserialize" type="verbatim">
         c->i.size = 1;
  </hack>
</func>
```

Step 1: Erlang side calls the port

```
fwrite(Port, Ptr, Filep) when port(Port) ->
    {PtrBinOrList, PtrLen} = serialize_contiguously(Ptr, 0),
    B = [ <<?EXAMPLE_FWRITE,        % Function call number
            PtrLen:32/integer>>,    % Arg 1: list length
          PtrBinOrList,             % Arg 1: I/O list
          << Filep:32/integer>> ], % Arg 2
    case catch erlang:port_command(Port, B) of
        true -> get_port_reply(Port);
        Err  -> throw(Err)
    end.
```

Aside: Examining the "callstate" Structure

```
typedef struct callstate {
  void                  (*invoke)(void *);
  int                   refc_bincount;
  ErlDrvBinary          *refc_bin[MAX_BINVS];
  struct {
      unsigned long     __stash[3];
      FILE *            filep;
      char *            ptr;
      size_t            size;
  } i;
  struct {
      size_t            ret_size_t;
  } o;
} callstate_t;
```

Step 2: C Side Deserializes Arguments

```
case EXAMPLE_FWRITE:
  c->invoke = invoke_s1_fwrite;
  EV_GET_UINT32(ev, &binlen, &p, &q);
  c->i.__stash[0] = binlen;
  c->i.ptr = (char *) EV_GETPOS(ev, p, q);
          if (c->refc_bincount == MAX_BINVS)
              goto error;
          ev->binv[q]->refc++;
          c->refc_bin[c->refc_bincount++] = ev->binv[q];
  if (edtk_ev_forward_N(ev, binlen, &p, &q, 1) < 0)
      goto error;
  EV_GET_UINT32(ev, &c->i.filep, &p, &q);
  c->i.size = 1; /* <hack place="post-deserialize" */
  break;
```

Step 3: Executing the Extension Function

```
static void
invoke_s1_fwrite(void *data)
{
  callstate_t *c = (callstate_t *) data;

  c->o.ret_size_t = fwrite(
                      c->i.ptr,
                      c->i.size,
                      c->i.__stash[0], /* nmemb */
                      c->i.filep
                    );
}
```

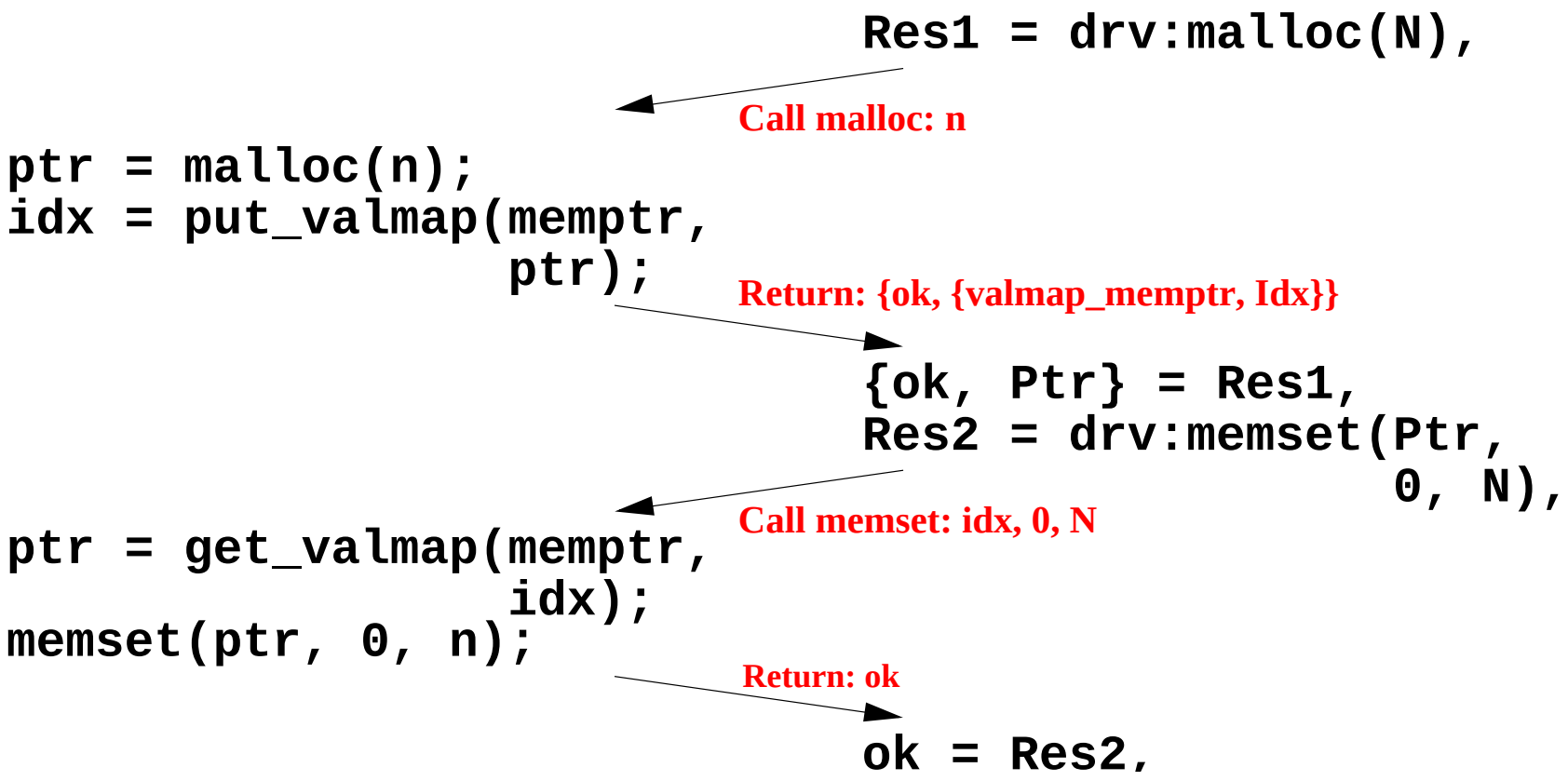
Step 4: Driver Returns Data to Erlang

```
static void
s1_ready_async(...)
{
  /* Omitted: decrement our binary refcounts */
  switch (c->cmd) {
    case EXAMPLE_FWRITE:
      reply_ok_num(desc, c->o.ret_size_t);
      break;
    default:
      edtk_debug("%s: bogus command %d", __FUNCTION__, c->cmd);
      break;
  }
  sys_free(c);
}
```

Step 5: Erlang Receives Term from Driver

```
get_port_reply(Port) when port(Port) ->
  receive
    {Port, ok} -> ok;
    {Port, ok, M} -> {ok, M};
    {Port, error, Reason} -> {error, Reason};
    %% Pipe driver messages
    {Port, {data, Bytes}} -> pipedrv_deser(Port, Bytes);
    {'EXIT', Port, Reason} -> throw({port_error, Reason});
    {Port, Reason} -> throw({port_error, Reason})
  end.
```

“Value Map” Example



Real Libraries and Applications

- simple1_drv: unit and regression testing, libc, system calls
- Libnet: portable "raw" packet transmission
- libpcap: portable "raw" packet receiving
- ethbridge: a multi-segment Ethernet bridge
- Berkeley DB: an embedded key-value database
- Spread: a reliable multicast library

Future Work

- Better documentation, more example drivers
- To SWIG or not to SWIG?
- Improve serialization: floats, bignums
- Generalize to handle "real" devices, networks
- Support C++
- Enable linked-in drivers to call Erlang functions