NRE: 8.6's non-recursive engine

Miguel Sofer

Tcl2008
Manassas, VA
State of a computation in 8.5

- **C-stack** (1 per thread)
  - Nesting of C-function calls
- **CallFrame stack** (1 per interp)
  - Nesting of [proc], [apply], [namespace eval]; this is what [upvar] and [uplevel] navigate
- **Tcl evaluation stack** (1 per execEnv/interp)
  - Bytecode engine stack and workspace
  - Special allocator supported (TclStackAlloc)
- **CmdFrame stack** (1 per interp)
The C stack in 8.5  (simplified)

A simple script

```tcl
proc foo {} {
    set x 1
    moo
    set x 2
}
proc moo {} {
    set x 1
    puts $x;# WE ARE HERE
    set x 2
}
foo
```

Stack Trace (simplified)

1: Tcl_PutsObjCmd
2: Tcl_EvalObjv [puts $x]
3: TclExecuteByteCode [moo]
4: TclObjInterpProc
5: Tcl_EvalObjv [moo]
6: TclExecuteByteCode [foo]
7: TclObjInterpProc
8: Tcl_EvalObjv [foo]
The C stack in 8.5 (the real thing)
The C stack in 8.5 (a problem?)

Stack overflow and recursion limits ...
- Platforms with small *default* stack (BSD and other unixes)
- Stack-hungry platforms (Win debug builds)
- Severely stack constrained platforms (Cisco, phones?)
- mod-8-3-branch version to reduce stack consumption
A simple script

```tcl
proc foo {} {
    set x 1
    moo
    set x 2
}
proc moo {} {
    set x 1
    puts $x;# WE ARE HERE
    set x 2
}
foo
```

Stack Trace (simplified)

1: Tcl_PutsObjCmd
2: Tcl_EvalObjv [puts $x]
3: TclExecuteByteCode [moo]
   Tcl_ObjInterpProc
   Tcl_EvalObjv [moo]
   TclExecuteByteCode [foo]
   Tcl_ObjInterpProc
4: Tcl_EvalObjv [foo]
The C stack in 8.6 (the real thing)
Full compatibility (*)

- Only additions to the API, no changes
- Modification of struct Command, changes to Tcl_[GS]etCommandInfo designed not to break extensions that call *objProc directly
- Changed structs (extended): Interp, Command, ExecEnv
The functions that are provided for the extension writer are:

- `Tcl_NRCreateCommand`
- `Tcl_NREvalObj`
- `Tcl_NREvalObjv`
- `Tcl_NRCmdSwap`
- `Tcl_NRAddCallback`

An NRE-enabled command requires two implementations: the regular `objProc` and a new `nreProc`. NRE provides a utility function `Tcl_NRCallObjProc` to make this task essentially trivial.
How to exploit NRE

A command currently implemented as

```c
int
MyCmdObjProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int objc,       /* Number of arguments. */
    Tcl_Obj *const objv[]) /* Argument objects. */
{
    <preparation>
    result = Tcl_EvalObjEx(interp, objPtr, flags);
    <postprocessing>
    <cleanup>
    return result;
}
Tcl_CreateObjCommand(interp, name, MyCmdObjProc, clientData, deleteProc);
```

... would look after adaptation as

```c
int
MyCmdNreProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int objc,       /* Number of arguments. */
    Tcl_Obj *const objv[]) /* Argument objects. */
{
    <preparation>
    Tcl_NRAddCallback(interp, MyPostProc, data0, data1, NULL, NULL);
    return Tcl_NR EvalObj(interp, objPtr, flags);
}
```

```c
int
MyPostProc(
    ClientData data[],
    Tcl_Interp *interp,
    int result)
{
    Tcl_Obj *fooPtr = data[0]; /* data0 retrieved */
    MyStruct *fooPtr = data[1]; /* data1 retrieved */
    <postprocessing>
    <cleanup>
    return result;
}
```

```c
MyCmdObjProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int objc,       /* Number of arguments. */
    Tcl_Obj *const objv[]) /* Argument objects. */
{
    return Tcl_NRCallObjProc(interp, MyCmdNreProc, clientData, objc, objv);
}
```

Tcl_NRCreateCommand(interp, name, MyCmdObjProc, MyCmdNreProc, clientData, deleteProc);
NRE-enabled core commands

- procs
- lambdas
- same-interp aliases
- namespace imports, ensembles (snit!)
- [eval], [uplevel], [namespace eval]
- TclOO methods
- [if], [for], [while], [foreach]
Non-enabled commands (yet?)

Mainly commands that change the execEnv
- [interp eval], [slave eval]
- Coroutines

Also
- [source]
- ...

From the C side: no NR-API for Tcl_Eval(), Tcl_EvalEx()
How does this happen? Concept

- **Trampolining:** do not *invoke* a function, rather ask your *caller* to do it for you (and then return)
- **Callbacks**
- **TEBC:** freeze a ByteCode, start running another one

**Stack Trace** (simplified)

1: Tcl_PutsObjCmd
2: Tcl_EvalObjv [puts $x]
3: TclExecuteByteCode [moo]
   TclObjInterpProc
   Tcl_EvalObjv [moo]
   TclExecuteByteCode [foo]
   TclObjInterpProc
4: Tcl_EvalObjv [foo]
New stack for *callbacks*: maintained by the ExecEnv

The workhorse is **TclNRRunCallbacks** (the trampoline)

```c
int TclNRRunCallbacks (  
    Tcl_Interp * interp,  
    int result,  
    struct TEOV_callback * rootPtr,  
    int tebcCall);
```

- loops running the EE's top callback
- Passes a callback's result to the next callback
- allows TEBC calls to ”leak through” (when called from TEBC) (*)
Callbacks

- Callback API:

```c
typedef int (Tcl_NRPostProc) (  
    ClientData data[],  
    Tcl_Interp *interp,  
    int result  
);
```

- Callback struct

```c
typedef struct TEOV_callback {
    Tcl_NRPostProc *procPtr;
    ClientData data[4];
    struct TEOV_callback *nextPtr;
} TEOV_callback;
```
TEBC has many (100?) local variables, but ...

typedef struct BottomData {
    struct BottomData *prevBottomPtr;
    TEOV_callback *rootPtr; /* State when this bytecode execution */
    ByteCode *codePtr; /* began; constant until it returns */
    /*! ---------------------------------------------------------- */
    TEOV_callback *atExitPtr; /* used on return FROM here */
    /*! ---------------------------------------------------------- */
    unsigned char *pc; /* These fields are used on return TO */
    ptrdiff_t *catchTop; /* this level: they record the state */
    int cleanup; /* when a new codePtr was received */
    Tcl_Obj *auxObjList; /* for NR execution */
} BottomData;
State of a computation in 8.6a3

- **C-stack** (1 per thread) *(lightly loaded)*
- **CallFrame stack** (1 per interp)
- **Tcl evaluation stack** (1 per execEnv), includes **NEW** BottomData stack.
- **CmdFrame stack** (1 per interp)
- **NEW CallbackStack** (1 per execEnv)

There are definite opportunities here for simplification and merging ...
NRE opens a world of new possibilities:

- Once TEBC knows how to freeze an execution ... put it aside and save it for later ⇒ coroutines
- Edit the Callback stack, rearrange the order of computations ⇒ proper tailcalls
- Schedule new evals from callbacks ⇒ CPS?
- ??? ⇒ ???
Miscelanea

- Not optimized – should get better during beta
- Some of the internal things may yet change
- C API not frozen ... TIP not yet submitted to a vote
- TEBC currently looks like a bomb exploded in it
- Debugging NRE is no fun; will have to invent something