TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

How TCL break, puts & help messages are handled by Lmod

Robert McLay

January 10, 2023

Outline

- Review of how TCL modulefiles are evaluated
- How .version and .modulerc file are evaluated
- Support for bare TCL break (LmodBreak())
- Support for TCL's puts
- Capturing help message from TCL modulefiles

How Lmod handles TCL modulefiles

- Use tcl2lua.tcl to read the modulefile.
- It evaluates all pure TCL code
- It outputs Lua strings for all module commands (setenv, etc)
- Lmod evalutes Lua output from tcl2lua.tcl
- Means that all TCL if stmts are evaluated by tcl2lua.tcl

How Lmod handles TCL modulefiles (II)

- Remember that tcl2lua.tcl is a separate code written in TCL
- It doesn't have access to the internal Lmod structures
- There is only a command-line interface between the two programs.

When things go awry

- Suppose you have TCL modules **Centos** and **B**

```
Centos::

    #%Module
    setenv SYSTEM_NAME Centos

And B::

    #%Module
    module load Centos

    if { $env(SYSTEM_NAME) == "Centos" } {
       # do something
    }
```

Converting the TCL B into Lua

```
load("Centos")
LmodError("can't read ënv(SYSTEM_NAME)̈: no such variable")
```

- Trouble: the TCL **load** command $\Rightarrow$ `load("Centos")`
- Cannot get the TCL load command to be evaluated before the TCL if block

How .version & .modulerc are eval'ed

- Lmod uses the RC2lua.tcl script to convert to Lua
- It only knows `module-version`, `module-alias`, ...
- It doesn't know about `setenv`
- I don't know what setenv means here

How Lmod implements TCL break

```
set a 10
while {$a < 20 } {
   puts "value of a: $a"
   incr acl2
   if { $a > 15} {
      break
   }
}
```

- Normal use: exit from loop.
- A bare TCL break is normally an error
- Lmod (and Tmod) stops evaluating current module.
- Lmod keeps all previous module evaluations intact
- Lmod continues evaluating after break
- Not sure what Tmod3 and Tmod do w.r.t. break

Examples

- `module load A B brkModule D`
- modules A and B are still loaded
- brkModule is essentially ignored
- D is loaded.
- `module load A B errModule D`
- Lmod internally loads A & B
- Loading errModule fails
- No new modules loaded.

How Lmod supports break

- Special code in tcl2lua.tcl to handle a bare break
- Lmod has to recover from a rejected modulefile

How tcl2lua.tcl handles bare break

```
set errorVal [interp eval $child {
    set returnVal 0
    ...
    set sourceFailed [catch {source $ModulesCurrentModulefile } errorMsg]
    if { $g_help ...} {
      ...
    }
    if {$sourceFailed} {
      if { $sourceFailed == 3 || $errorMsg == invoked "break" outside of a loop} {
          set returnVal 1
          myBreak               # output "LmodBreak into Global
          showResults           # Write output
          return $returnVal     # return with error status
       }
       reportError $errorMsg    # output error message
       set returnVal 1          # return with error status
    }
    showResults                  # Write output for normal translation
    return $returnVal            # return with OK status
}]
```

- A bare break is an error in TCL
- tcl2lua.tcl captures that
- generates ”LmodBreak()”

How Lmod handles LmodBreak()

- Lmod maintains a stack of module “states”
- It is called “FrameStk“
- It contains:
  1. VarT: new env vars values
  2. ModuleTable: The currently loaded modules
  3. mname: Current module object to be loaded.
- Support for FrameStk was added with Lmod 7 rewrite
- Correct support for Break was added in 8.7+

FrameStk action during module loads

- Each module load creates a new FrameStk entry
- Currently loaded module succeeds ⇒ overwrites previous entry
- Break causes the current entry to be thrown away

Another Break example

```
$ cat StdEnv.lua
load("A")
load("B")
load("BRK")
load("D")

$ ml StdEnv; ml
Currently loaded modules:
  1) A   2) B  3) D
```

- The contents of the BRK module are ignored

Handling TCL puts

- TCL `puts` $\Rightarrow$ calls `myPuts` thru child interpreter
- puts and myPuts takes upto 3 arguments
- It took years to get this correct
- `myPuts` write to a global array in tcl2lua.tcl
- the `showResults` sends it to stdout for lua to evaluate
- Message sent to stderr use LmodMsgRaw() function

myPuts arguments

- puts can only have 1 to 3 arguments
- puts <-nonewline> <channel> msg
- puts msg ⇒ writes to stdout (at end)
- puts stdout msg ⇒ writes to stdout (at end)
- puts stderr msg ⇒ writes to stderr
- puts prestdout msg ⇒ writes to stdout but at the beginning of output

Handling TCL help messages

```
proc ModulesHelp
    puts stderr "The TACC Amber installation ..."


Lmod wants:
help ([===[The TACC Amber installation ...]===])
```

- Converting TCL help message was tricky
- tcl2lua.tcl has to capture the output when executing ModulesHelp
- myPuts has a special mode when running ModulesHelp

```
if { $g_help && [info procs "ModulesHelp"] == "ModulesHelp" } {
    set start "help(\[===\["
    set end   "\]===\])"
    setPutMode "inHelp"
    myPuts stdout $start
    catch  ModulesHelp  errMsg
    myPuts stdout $end
    setPutMode "normal"
}
```

- in “inHelp” mode output to stderr is written to stdout

Help Conversion Example

```
help([===[
The TACC Amber installation only includes the parallel Sander/pmemd modules.
The Amber modulefile defines the following environment variables: ...

Version 9
]===])
```

- This way help message work the same with Lua and TCL modulefiles

Conclusions

- TCL to Lua conversion works well
- But it is NOT perfect.
- TCL Break, puts and help message required special foo

Next Time

- How to use `check_module_tree_syntax`

Future Topics

- I am on vacation (a.k.a holiday) in early February
- Next Meeting will be Feb 14th