Providing Current Module Data to Hooks

▶ Here we explore some core concepts in how Lmod works

▶ The FrameStk: A stack of modules in the load process

▶ The ModuleTable(MT): The currently loaded modules ($\neq$ frameStk)

▶ MName: ModuleName objects

Core Lmod Concept: sn, fullName, version

- ▶ The FullName of the module is the shortname/version
- ▶ N/V: FullName: gcc/9.4.0: sn: gcc, version: 9.4.0
- ▶ C/N/V: FullName: cmplr/gcc/11.2, sn: cmplr/gcc, version: 11.2
- ▶ N/V/V: FullName: bowtie/64/22.1, sn: bowtie, version: 64/22.1
- ▶ N/V/V: FullName: bio/bowtie/64/22.1, sn: bio/bowtie, version: 64/22.1

Core concept: Singleton

► The Lmod code uses the Design Pattern: Singleton
► A singleton will build an object only once
► No matter how many times it is asked for

MName: userName ⇒ sn, fullName

- `module load` *foo*
- *foo* is the userName
- The userName might be an sn or a fullName
- Or somewhere in between for N/V/V
- The MName class builds an mname object that knows sn, fullName, etc

FrameStk

- ► Named borrowed from StackFrame
- ► It is a stack of mname object that are in the process of being loaded
- ► It also contains the current ModuleTable

ModuleTable A.K.A. mt

► This is a hashtable or dict of the currently loaded modules
► This table is stored in the user environment to store the state
► This is how `module load` can work
► The MT is split into 256 character blocks when saved in user's environment
► And stored as $_ModuleTable001_ ...
► The module properties are stored in mt.

ml −mt

```
_ModuleTable_ = {
  MTversion = 3,
  mT = {
   mkl = {
      fn = "/opt/apps/modulefiles/Core/mkl/mkl.lua",
      fullName = "mkl/mkl",
      loadOrder = 1,
      propT = {
        arch = {
          gpu = 1,
        },
      },
      stackDepth = 0,
      status = "active",
      userName = "mkl",
      wV = "*mkl.*zfinal",
    },
  },
},
```

Steps to load a module

1. Convert userName to mname object
2. Push mname object onto frameStk stack.
3. Get current mt from frameStk
4. Add mname to mt and mark as pending
5. Load current modulefile by evaluating as a lua program
6. If no errors then change status in mt to active for current module.
7. Pop top entry from frameStk

How can size(frameStk) > 1?

- ▶ A modulefile can load other modulefiles
- ▶ This stack is only as deep as there are pending modules
- ▶ Direct user loads have a stack size of 1.
- ▶ Dependent loads will have a stack size > 1
- ▶ Some sites use this for module tracking
- ▶ They only record a modulefile if the stack size is 1.

How to get current module data in a hook

- Pay-off slide
- Ask for frameStk object from the singleton
- Ask for the current mname object from frameStk
- Ask for the current mt object from frameStk
- Ask for the sn from mname
- Ask mt:haveProperty(sn, propname, propvalue)

ml –mt

```
function M.isVisible(self, modT)
   local frameStk   = require("FrameStk"):singleton()
   local mname      = frameStk:mname()
   local mt         = frameStk:mt()
   local mpathA     = mt:modulePathA()
   local name       = modT.fullName
   ...

   modT.isVisible = isVisible
   modT.mname     = mname
   modT.sn        = mname:sn()
   modT.mt        = mt
   hook.apply("isVisibleHook", modT)
   return modT.isVisible
end
```

Side notes

► Note that the mt table is key'ed by sn
► This is why Lmod has the one name rule.
► It is really the one sn rule.

Next Topic

- Lmod Testing System
- Monday March 1st at 15:30 UTC (9:30am US Central)

TACC

Future Topics

► Lmod Testing System?
► More internals of Lmod?
► Guest Presentation of special issues?