Curso de Introdução Prática ao Simulador de Redes NS-2

Instrutor
Eduardo da Silva
Monitores
Elisa Mannes
Fernando H. Gielow
Urlan S. de Barros
Coordenador
Prof. Aldri L. dos Santos

Outubro de 2009

Roteiro

- Discussão dos exercícios anteriores
- Integração C++/oTCL
- Um exemplo completo de um código
- Novos exercícios

Estrutura do TclCl

Várias classes são definidas em */tclcl*

- **Tcl** → Métodos que o C++ usará para acessar o intepretador
- **TclObject** → Classe base para todos os objetos do simulador que são espelhados na hierarquia compilada
- **TclClass** → Define a hierarquia da classe interpretada, e os métodos para permitir ao usuário instanciar TclObjects
- **TclCommand** → Usada para definir comandos do interpretador
- **EmbeddedTcl** → Comandos internos de alto nível que facilitam a configuração das simulações
- **InstVar** → Métodos de acesso aos atributos C++ como variáveis no oTcl

Classe TclClass

- Uma classe virtual pura
- Classes derivadas dela fornecem duas funções:
  - constroem a hierarquia da classe interpretada para espelhar a hierarquia da classe compilada
  - fornecem métodos para instanciar novos TclObjects

Exemplo

```
static class PingClass : public TclClass {
public:
  PingClass() : TclClass("Agent/Ping") {}
  TclObject* create(int, const char*const*) {
    return (new PingAgent());
  }
} class_ping;
```

Classe TclObject

- É a classe base da maioria das outras classes
- Todo objeto dessa classe é criado a partir do interpretador
- A TclClass faz a associação entre esses objetos e os objetos da hierarquia compilada

Classe TclObject

Bindings

- Estabelece uma associação bi-direcional
- Atributos interpretados e compilados acessam o mesmo dado

Exemplo em oTcl

```
set ping [new Agent/Ping]
$ping set packetSize_ 500
```

Exemplo em C++

```
PingAgent::PingAgent() : Agent(PT_PING)
{
  bind("packetSize_", &size_);
  bind("off_ping_", &off_ping_);
}
```

Classe TclObject

Commands

- Implementa métodos do OTcl em C++
- Chamada oTcl
  - $ping send

Exemplo em C++

```
int PingAgent::command(int argc, const char*const* argv) {
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      Packet* pkt = allocpkt();
      hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
      hdr->ret = 0;
      hdr->send_time = Scheduler::instance().clock();
      send(pkt, 0);
      return (TCL_OK);
    }
  }
  return (Agent::command(argc, argv));
}
```

Classe Tcl

- Possui a referência para o interpretador Tcl
  - chamar procedimentos em OTcl
  - obter resultados de expressões em OTcl
  - passar um resultado para o OTcl
  - retornar um código de sucesso/falha para o Otcl
  - armazenar e realizar procuras por referências a TclObjects
- Antes de acessar qualquer método do intepretador, usar o comando:
  - Tcl& tcl = Tcl::instance();

Classe Tcl

Comandos para o interpretador

- Tcl::eval(char *): passa a string para o intepretador
- Tcl::eval(): supõe que o comando já está em tcl.buffer
- Tcl::evalf(char *, par1, par2, ...): "*printf like*"

Exemplo

```
void PingAgent::recv(Packet* pkt, Handler*) {
 ...
 char out[100];
 sprintf(out, "%s␣recv␣%d␣%3.1f", name(),
       hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
       (Scheduler::instance().clock()-hdr->send_time) * 1000);
 Tcl& tcl = Tcl::instance();
 tcl.eval(out);
 ...
}
```

Um protótipo de Ping

- Será estudado o Ping
- Código que acompanha a instalação padrão do NS-2
- Desenvolvido por Marc Greiss

O arquivo *ping.h*

```
#ifndef ns_ping_h
#define ns_ping_h
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"

struct hdr_ping {
  char ret;
  double send_time;
};
class PingAgent : public Agent {
 public:
  PingAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  int off_ping_;
};
#endif
```

O arquivo *ping.cc* I

```
#include "ping.h"

static class PingHeaderClass : public PacketHeaderClass {
public:
  PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
sizeof(hdr_ping)) {}
} class_pinghdr;

static class PingClass : public TclClass {
public:
  PingClass() : TclClass("Agent/Ping") {}
  TclObject* create(int, const char*const*) {
    return (new PingAgent());
  }
} class_ping;

PingAgent::PingAgent() : Agent(PT_PING) {
 bind("packetSize_", &size_);
 bind("off_ping_", &off_ping_);
}
```

O arquivo *ping.cc* II

```
int PingAgent::command(int argc, const char*const* argv) {
 if (argc == 2) {
  if (strcmp(argv[1], "send") == 0) {
   Packet* pkt = allocpkt();
   hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
   hdr->ret = 0;
   hdr->send_time = Scheduler::instance().clock();
   send(pkt, 0);
   return (TCL_OK);
  }
 }
 return (Agent::command(argc, argv));
}
```

O arquivo *ping.cc* III

```
void PingAgent::recv(Packet* pkt, Handler*) {
 hdr_ip* hdrip = (hdr_ip*)pkt->access(off_ip_);
 hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
 if (hdr->ret == 0) {
  double stime = hdr->send_time;
  Packet::free(pkt);
  Packet* pktret = allocpkt();
  hdr_ping* hdrret = (hdr_ping*)pktret->access(off_ping_);
  hdrret->ret = 1;
  hdrret->send_time = stime;
  send(pktret, 0);
 } else {
  char out[100];
  sprintf(out, "%s␣recv␣%d␣%3.1f", name(),
     hdrip->src_ >> Address::instance().NodeShift_[1],
     (Scheduler::instance().clock()-hdr->send_time) * 1000);
  Tcl& tcl = Tcl::instance();
  tcl.eval(out);
  Packet::free(pkt);
 }
}
```

Outras alterações I

Arquivo *ns/common/packet.h*

```
enum packet_t {
  ....
  // insert new packet types here
 PT_PING,
 PT_NTYPE // This MUST be the LAST one
};
....
# No mesmo arquivo
....
class p_info {
public:
 p_info() {
   ....
  name_[PT_PING]="Ping";
  name_[PT_NTYPE]= "undefined";
 }
  ....
}
```

Outras alterações II

Arquivo *ns/tcl/lib/ns-default.tcl*

```
Agent/Ping set packetSize_ 64
```

Arquivo *ns/tcl/lib/ns-packet.tcl*

```
foreach prot {
  ...
  Ping
  ...
}
```

Outras alterações III

Arquivo *ns/Makefile*

Finalmente, a alteração final

```
OBJ_CC = \
...
apps/ping.o ,\
...
```

O script TCL

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$ns duplex-link $n0 $n1 \
   1Mb 10ms DropTail
$ns duplex-link $n1 $n2 \
   1Mb 10ms DropTail

set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
$ns attach-agent $n2 $p1
```

```
Agent/Ping instproc recv {from rtt} {
 self instvar node_
 puts "node [$node_ id] \
    received ping answer from \
    $from with round-trip-time $rtt ms"
}

$ns connect $p0 $p1

$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.6 "$p0 send"
$ns at 0.6 "$p1 send"
$ns at 1.0 "finish"

$ns run
```

Exercício 1

Fazer um agente que funcione da seguinte forma:

- um agente, que se encontra no nó A, por exemplo, pode enviar dois tipos de mensagens diferentes para o outro nó, por exemplo B
- esse agente do nó B, ao receber a mensagem, verifica a mensagem e manda uma resposta, que depende do tipo da mensagem recebida
- todas as mensagens recebidas devem ser gravadas no arquivo de *trace* sendo que os primeiros bytes devem informar o tipo da mensagem (envio ou resposta) e o código dela (1 ou 2)

Obs. Essas mensagens podem ser fixas, tipo MENSAGEM1, MENSAGEM2 e RESPOSTA1, RESPOSTA2

Exercício 2

Alterar o código dos programas anteriores:

- Permitir que o destinatário possa ser informado como parâmetro do comando **send**
- Isso elimina a necessidade de ter que “conectar” os dois agentes previamente
- *Detalhe*: isso é bem mais difícil do que parece :)

OBRIGADO!