Secure Operating System Design and Implementation
Remote Procedure Call

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

February 1, 2011

Part I

Remote Procedure Call (RPC) overview

Overview

Remote Procedure Call is

- a procedure call mechanism
- the procedure call invoker is the caller
- the procedure execution is the callee
- callee and caller are in different address spaces
- any value from caller needed by the callee must be an RPC parameter
- Consider `add(a, b)`
  - a and b are sent from caller to callee
  - callee performs $a + b$, sends result to caller
  - caller read results and returns it as the result of add

RPC structure

The RPC mechanism hides most of the details of inter-address space communication

- Writing the values out (called marshalling) by the caller
- Reading the value in (called unmarshalling) by the callee
- At the caller side, a stub is invoked (e.g., `add`)
- At the callee side, a skeleton is invoked, which ultimately invokes the remote procedure (e.g., `add`)
- The stub and skeleton is generated by an RPC compiler
- The input to the RPC compiler is the declaration of its procedures (and possibly data)

Marshalling/unmarshalling issues

marshalling and unmarshalling deals with:

data size parameters can have different size

data type parameters are typed

packet size there may be a limitation of the amount of data which is sent as a unit.

in memory differences between “in memory” layout at the caller and callee

on-the-wire differences between “in memory” and on-the-wire layout.

RPC advantages

modularity RPC implementation independent of remainder of code

documentation RPC interface clearly documented

modifiable easy to change the RPC interface

optimization RPC implementation can be changed w/o changing application

testing Easy to test RPC interface separately from its use in applications

consistency skeleton and stub match since generated from same specification

Part II

RPC marshalling/unmarshalling issues

Marshalling/unmarshalling issues

- The biggest problem occurs when callee and caller are on different computer architectures
- Then the in-memory layout will be different for many things at the callee and caller side
- This requires the stub and skeleton to compensate for it

Architecture type issues

size most easily compensated for by using architecture-independent sizes

alignment architecture may require some *n*-byte objects (e.g., ints) to start at an address divisible by *n*.

type In general, integer and character work well, but floating point does not. Char signedness may differ between architecture.

pointers values are always relative to an address space

Part III

Ethos RPC

Ethos RPC

- Ethos RPC is used to communicate between Ethos kernel and Dom0 shadow daemon
- Could be used in other places, for example:
  - IPC within Ethos
  - Networking between Ethos systems
  - And could be extended to work for Ethos's file system
- The direction we go in will depend on user space programming language
- But we are committed to maintaining types across address spaces

Asynchrony

- RPC calls may have arbitrary latency
- Thus, we would like to have multiple RPCs outstanding at a time
- Will therefor need to match call with response
- An ID is needed for that purpose
- Ethos RPC does not directly support asynchronous RPC, rather it uses one-way RPCs.

One-way RPCs

- Ethos only uses one-way RPCs.
- The RPC is from caller to callee with no return value.
- To build a two-way RPC
  - Create an ID for the RPC instance
  - Do a one-way call, passing the ID
  - Callee invokes a reply RPC, passing the ID
  - The ID is used to match up the RPC call with reply
- In Ethos, we're using the eventId as the RPC identifier

Parameters

Can be any of the following types

- integers: int32, int64, uint32, uint64
- characters: char8, uchar8
- vectors of primitive types: A vector has a size and a number of elements

On-the-wire

The virtual packet is of the following form

- The procedureId is a 32-bit unsigned integer
- Each parameter takes up an integral number of 4-byte slots
- Vectors have a 32-bit size plus an integral number of 4-byte slots

Part IV

Connections and real packets

Connections and real packets

- Each RPC flows over a connection
- connection are multiplexed over a tunnel, such as an Ethernet connection from Ethos to Dom0
- The connection specifies the end-points of the communication
- Over the connection, real packets (of a maximum size) flow
- Thus large RPCs may need to be split into multiple real packets

Virtual packets

- Virtual packets are limited in size only by memory
- Physical packets are reconstructed on the receiving end into a virtual packet
- The virtual packet is then unmarshalled

Part V

Conclusions

Conclusions

- RPC takes care of many low level protocol issues, enabling the programmer to focus on communication pattern.
- Ethos's RPC supports integers and characters and vectors of the same
- It supports only one-way RPC, since it is a good building block for asynchronous or synchronous RPC and is highly flexible
- The RPC mechanism also supports connections (i.e., multiple targets of the RPC).