RUSS Specification

Contents

Introduction

This document provides the specification for RUSS.

The goal of RUSS is to expose a collection of services addressable in a hierarchical namespace, each of which establish zero or more I/O streams for communication, and a means for providing an exit status when a service terminates. A service is called with an operation, a service path, an ordered list of zero or more strings, and an unordered map of zero or more key-value pairs. The fundamental client function is modelled after the `execve()` call.

RUSS is targeted to UNIX and UNIX-like systems which support domain sockets.

Definitions

Credentials

Credentials uniquely identify a principal (e.g., user) on a host. Credentials are not transferable to non-local hosts.

Deadline

A deadline is a fixed point in time local to a host. The time specified by a deadline does not change. This is in contrast to a timeout which is defined relative to a base time. A deadline is not transferable to non-local hosts.

Exit Status

An integer value in the range 0-255. The exit status is typically used to indicate success (0) or failure/non-success (non-0).

Operation

A service supports one or more operations (e.g., list, execute).

Ordered Argument List

A list of strings whose order is significant.

Service Path

All services are identified by a /-separated list of strings comprising a service path. The service path is prefixed with a socket address, after which the path components are interpreted by a server in search of a matching service.

Socket Address

The access point for all services is a domain socket address. It may be dedicated to a server which provides a collection of services or as an indirect access point to one or more such servers.

Stream

An object for uni or bidirectional communication. Unless otherwise stated, all services provide and communicate over stdin, stdout, and stderr I/O streams. Additional streams may be provided.

Unordered Map

A collection of key-value pairs (strings) whose order is not significant.

Wait Return Value

The value returned by an API wait call that is performed to get an exit status. The value may indicate success, failure, or some other status.

Constants

Deadlines

Value	Constant	Description
`INT64_MAX` (at least)	`RUSS_DEADLINE_NEVER`	Some future time that will never arrive given the local host clock calls.

Exit Statuses

Value	Constant	Description
0	`RUSS_EXIT_SUCCESS`	Success.
1	`RUSS_EXIT_FAILURE`	Generic failure.
126	`RUSS_EXIT_CALLFAILURE`	Dial call failure.
127	`RUSS_EXIT_SYSFAILURE`	System failure, e.g., no exit value returned.

Messages

A common set of messages are used to indicate an noteworthy condition as sent over the stderr connection file descriptor:

Constant	Description
`RUSS_MSG_BADARGS`	Bad or missing arguments for service call.
`RUSS_MSG_BADCONNEVENT`	Unexpected connection event.
`RUSS_MSG_BADOP`	Unsupported operation.
`RUSS_MSG_BADSITUATION`	Unexpected situation.
`RUSS_MSG_NOACCESS`	Insufficient privilege.
`RUSS_MSG_NODIAL`	Could not dial service.

RUSS_MSG_NOEXIT	No exit status.
RUSS_MSG_NOLIST	Service list not available.
RUSS_MSG_NOSERVICE	Service request not handled.
RUSS_MSG_NOSWITCHUSER	Service could not switch user.
RUSS_MSG_UNDEFSERVICE	Invalid service path.

Operations

Constant	Value	Alternative String	Description
RUSS_OPNUM_NOTSET	0		Should never be this for an accepted request.
RUSS_OPNUM_EXTENSION	1		If operation string is unrecognized. Useful for future operations.
RUSS_OPNUM_EXECUTE	2	execute	Execute a service at a given service path.
RUSS_OPNUM_HELP	3	help	Provide server or service-specific help as text.
RUSS_OPNUM_ID	4	id	Provide identifying information about the the server.
RUSS_OPNUM_INFO	5	info	Reserved.
RUSS_OPNUM_LIST	6	list	Provide a newline separated list of services available below a given service path.

Additional operations may be defined in the future.

Wait Return Values

Constant	Value	Description
RUSS_WAIT_UNSET	1	Unset; should not occur.
RUSS_WAIT_OK	0	Success.
RUSS_WAIT_FAILURE	-1	Generic failure.
RUSS_WAIT_BADFD	-2	Exit fd is closed/invalid.
RUSS_WAIT_TIMEOUT	-3	Wait call timed out.

Data Structures

Data structures that are fundamental to RUSS. They are presented as C `structs`.

Request

```
struct russ_req {
    char        *protocolstring;    /**< identifies russ protocol */
    char        *op;        /**< operation string */
    russ_opnum  opnum;        /**< operation number*/
    char        *spath;        /**< service path */
    char        **attrv;    /**< NULL-terminated array of attributes (as name=value strings) */
    char        **argv;        /**< NULL-terminated array of args */
};
```

Client Connection

```
struct russ_cconn {
    int            sd;        /**< socket descriptor */
    int            fds[RUSS_CONN_NFDS];        /**< array of fds */
    int            sysfds[RUSS_CONN_NSYSFDS];    /**< array of system fds */
};
```

Server Connection

```
struct russ_sconn {
    struct russ_creds    creds;        /**< credentials */
    int            sd;        /**< socket descriptor */
    int            fds[RUSS_CONN_NFDS];        /**< array of fds */
    int            sysfds[RUSS_CONN_NSYSFDS];    /**< array of system fds */
};
```

Credentials

```
struct russ_creds {
    long    pid;
    long    uid;
    long    gid;
};
```

Protocol

The following describes the protocol corresponding to the protocol string (`RUSS_PROTOCOLSTRING`) 0010.

Encodings

Object	Composite	Description
`byte`		• 1 byte; full 8 bits
`bytes`	`size:int32` `payload:n*byte`	• size-encoded byte array
`int32`		• unsigned integer • 4 bytes • big-endian
`sarray0`	`size:int32` `payload:n*string`	• NULL-terminated string array • NULL not encoded
`string`	`size:int32` `payload:n*byte`	• NULL-terminated C string • NULL character is encoded

Connection

Comprises all that is required to establish a viable connection between client and server which amounts to a set of stream descriptors sent by the server and received by the client.

Steps

Phase	Client	Server
1	• connect to socket address	

2		• accept socket connect request
3	• encode request • send request to server	
4		• decode request • get client credentials from socket • lookup service handler • call service handler • set up I/O streams • encode response • send response • close socket
5	• decode response • close socket	

Once the socket connection has served its purpose, it is no longer used. However, the I/O streams and exit descriptors are a part of the RUSS connection and remain available until closed by either of the sides.

Request

The request is sent over the socket.

```
size:int32
protocolstring:string
future:bytes
spath:string
op:string
attrv:sarray0
argv:sarray0
```

Where `size` is the total number of bytes of the request excluding the `size`.

Note: The client credentials are not part of the request message but obtained from the socket.

Response

The response is sent over the socket.

```
nfds:int32
statuses:int32+n*byte
(nfds*fd using sendmsg())
```

Exit

Exiting comprises all that is required to inform the client that the service has completed along with the completion status.

The status of the non-exit streams is not defined. I.e., it is left to the service to deal with the streams, such as to leave them open or to close them.

Steps

Phase	Client	Server
1		• send exit value • close stream

2	• receive exit value • close stream	

Exit Value

The exit value is sent over the exit stream.

```
exitvalue:int32
```