ZeroVM channels description

Channel = [host file name], [guest device name], [access type], [limit for reads], [limit for read bytes], [limit for writes], [limit for write bytes]

ex.: Channel = /tmp/file.tmp, /dev/stderr, 0, 0, 0, 1048576, 1048576

Channel = [host device name], [guest device name], [access type], [limit for reads], [limit for read bytes], [limit for writes], [limit for write bytes]

ex.: Channel = /dev/stdin, /dev/stdin, 0, 1073741824, 1073741824, 0, 0

Channel = [host socket], [guest device name], [access type], [limit for reads], [limit for read bytes], [limit for writes], [limit for write bytes]

ex.: Channel = tcp:127.0.0.1:55431, /dev/in/some_host, 0, 1073741824, 1073741824, 0, 0

Channel = [host identifier], [guest device name], [access type], [limit for reads], [limit for read bytes], [limit for writes], [limit for write bytes]

ex.: Channel = tcp:21:, /dev/out/node21, 0, 0, 0, 1073741824, 1073741824

Channels are the key component of ZeroVM I/O subsystem

On the host system side the channels are backed by the file system using either regular files or tcp sockets.

On the guest side the channel is a device file. It could be used as either character

or block device it depends on the usage pattern and access type (see access types below).

All guest devices must reside in the /dev directory or its subdirectories.

Channel fields passed via the manifest to ZeroVM (comma delimited, see examples above):

name -- can be a local file or device name or tcp socket or host identifier (see more on host identifiers below)

alias -- device name for the user side. Must follow the device naming scheme: /dev/device-name

type -- access type. Can be in 0..3 range

gets -- limit for reads allowed from this channel

get_size -- limit on total amount of data to be read from this channel in bytes

puts -- limit for writes allowed for this channel

put_size -- limit on total amount of data to be written to this channel in bytes

Fields available for the untrusted code (see api.txt):

limits -- 4 limits for the channel

size -- channel size. only available for random access channels

type -- access type

name -- the channel name (the alias from the manifest)

the indices in the channels array can be used as "handles". also it is guaranteed

that handles 0..2 represent stdin, stdout and stderr

Channel access types:

0 - sequential read / sequential write (can be used as character device)

1 - random read / sequential write

2 - sequential read / random write

3 - random read / random write

For now ZeroVM supports 6 kinds of channels (defined by combination of access type and 4 read/write limits):

- sequential read only channel (access type 0, both write limits are 0)

- random read only channel (access type 3, both write limits are 0)

- sequential write only channel (access type 0, both read limits are 0)

- random write only channel (access type 3, both read limits are 0)

- appendable channel. random read, sequential write. if the channel is not empty

  the write position is set next to the last byte (access type 1, any limits)

- full random access channel. can be read or written from any valid position (access type 3, any limits)

There are 3 mandatory channels (standard c90 streams): stdin, stdout, stdout.

The channels have well known device names "/dev/stdin", "/dev/stdout" and "/dev/stderr".

You must have all 3 mandatory channels present in your manifest.

/dev/stdin is a sequential read only channel

ex: Channel = path, /dev/stdin, 0, reads, bytes, 0, 0

/dev/stdout and /dev/stderr are sequential write only channels

ex: Channel = path, /dev/stdout, 0, 0, 0, writes, bytes

Channel is a file abstraction over the local files and network streams. Local files

can have random access type, network streams are always sequential.

All channels are opened before the session start and closed after session end. In case of

the channels i/o error ZeroVM will not start. ZeroVM preallocates specified byte size

for the local writable channels (this can be changed with -P switch, see zerovm_switches.txt).

Network (socket based) channels

All socket based channels are either sequential read only or sequential write only.

They represent unidirectional connection between two ZeroVM instances.

You can have as many network channels as you want, although it's not advisable to have more

than one channel in the same direction between the same two ZeroVM instances.

All network channels use TCP sockets. Write only channel will listen on socket while

the read only channel will connect to the same socket from another instance.

All the read operations on the guest side will block until some data is available.

If you read a specific amount of data (using pread() for example) the read will block

until this amount of data is available.

There is no support for unblocking reads for ZeroVM channels, it's by design.

Example of bidirectional connection between two ZeroVM instances:

Instance #1, IP addr 10.0.0.1

Channel = tcp:10.0.0.1:34423, /dev/out/instance2, 0, 0, 0, 9999999, 9999999

Channel = tcp:10.0.0.2:56645, /dev/in/instance2, 0, 9999999, 9999999, 0, 0

Instance #2, IP addr 10.0.0.2

Channel = tcp:10.0.0.1:34423, /dev/in/instance1, 0, 9999999, 9999999, 0, 0

Channel = tcp:10.0.0.2:56645, /dev/out/instance1, 0, 0, 0, 9999999, 9999999

Each read from read only network channel can result in zvm_eof indicator read.

This means that the other party closed the channel, you will get the same zvm_eof

each time you try to read from this channel again.

If integrity checks are in place (zerovm was run with -e option)

zvm_eof will contain channel integrity checksum.

Host identifiers

In the case of clustered runs there is no way to know the network topology before actually trying to run ZeroVM instances.

Host IPs and ports are not known until the very start of ZeroVM executable.

To solve this problem we introduce the host identifiers.

Each ZeroVM instance (guest) in clustered configuration gets identifier (just a running number)

The identifier is unique only within the current cluster configuration.

For example if we want to run map-reduce job with 4 mappers and 5 reducers, there will be 9 identifiers (1..9)

and one number will be assigned for each instance (be it mapper or reducer).

To resolve the host identifiers you will need the following data in manifest:

NameServer = udp:[ip address]:[port]

NodeName = [instance name], [host identifier]

Name server line contains the ip address and port of udp name server that will resolve host ids.

Node name line contains the name of this node (just a string) and the host identifier of this node

Example:

NameServer = udp:10.0.0.254:5544

NodeName = mapper-21, 34

The two instance example (see above) will look like this:

Instance #1

Channel = tcp:2:, /dev/out/instance-2, 0, 0, 0, 9999999, 9999999

Channel = tcp:2:, /dev/in/instance-2, 0, 9999999, 9999999, 0, 0

NameServer = udp:10.0.0.254:5544

NodeName = instance-1, 1

Instance #2

Channel = tcp:1:, /dev/in/instance-1, 0, 9999999, 9999999, 0, 0

Channel = tcp:1:, /dev/out/instance-1, 0, 0, 0, 9999999, 9999999

NameServer = udp:10.0.0.254:5544

NodeName = instance-2, 2

If the manifest contains host identifiers ZeroVM will do the following on startup:

1) for all write only channels, bind on any available port (but 1st will try 49152..65535) for each one

2) send discovery packet to name server with all the bound ports

discovery packet also contains all identifiers for read only channels present in the manifest

3) wait for name server reply

4) name server replies with mapping of identifiers sent in 2) to ip:port tuples

5) connect to all read only channels

After this 5 step operation all channels will be set up with correct unidirectional data paths

The reference implementation of name server daemon can be found in ZRT library (name_server.py).

You can also use ZRT networked samples to see the flow, namely 'reqrep' or 'disort' sample.

To find more info about name server consult name_server.txt

Limitations

- The number of channels in the manifest is limited to 6548 "Channel" lines

- user program cannot bind and connect network channel to itself