# IPC Library for Co-Simulation

Version 0 Revision 1

Aug. 1, 2021 (July 1, 2021)

Ando Ki, Ph.D.
andoki@gmail.com

## Copyright

Copyright © 2021 Ando Ki
All rights are reserved by Ando Ki.

This contents and its associated materials are licensed with the 2-clause BSD license to make the program and library useful in open and closed source products independent of their licensing scheme. Each contributor holds copyright over their respective contribution.

All contents are provided as it is WITHOUT ANY WARRANTY and NO TECHNICAL SUPPORT will be provided for problems that might arise.

## Abstract

This document describes Inter-Process Communication (IPC) library for message exchange between two independent processes in full duplex fashion.

## Table of Contents

Copyright © 2021 Ando Ki

# 1 Introduction

Inter-Process Communication[1] (IPC) is a mechanism to exchange data between threads beyond the process boundaries, in which two threads does not share address space.

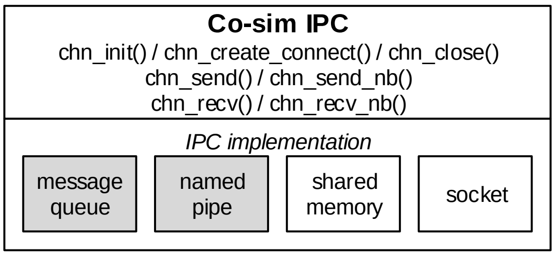
Co-sim IPC
chn_init() / chn_create_connect() / chn_close()
chn_send() / chn_send_nb()
chn_recv() / chn_recv_nb()
IPC implementation
message queue
named pipe
shared memory
socket

**Figure 1: Co-sim IPC library**

This library focuses on IPC between threads (or processes) residing on the same computer. It uses 'System V message queue' for Linux platform and 'named pipe' for Windows platform. This library supports multiple channels using logical channel identifications.

# 2 API

## 2.1 API convention

Each API returns 0 on successful completion. Otherwise, it returns non-zero number and error number is stored in internal variable.

## 2.2 initializing, creating, connecting and closing channel

This library maintains an array that keeps channel handler and all API functions use channel identification number instead of channel handler.

### 2.2.1 initialize: chan_init()

Function prototype:

```
int chn_init (void);
```

Return:

- 0: successful completion
- !=0: on failure

Synopsis: It initializes all global variables. This will be called automatically when 'chn_create_connect()' or 'chn_connect()' is called without initializing.

[1] shared memory, message, pipes, sockets, and so on

Copyright © 2021 Ando Ki

### 2.2.2 create: chn_create_connect()

Function prototype:

```
int chn_create_connect (int chan_id, int side);
```

Argument:

- int chan_id: channel identification number from 0 to MAX_NUM_CHAN[2]-1
- int side: one of ‘CHAN_HOST’ or ‘CHAN_TARET’
    - ‘CHAN_HOST’ has responsibility to maintain the channel.

Return:

- 0: channel handler in  valid value of pointer on successful completion
- !=0: on failure

Synopsis: It creates channel if there is no channel with the channel id and then connects to it. It connects the channel id without creating it. Each channel is full-duplex and each entity at the end of channel should know who it is.

### 2.2.3 connect: chn_connect()

Function prototype:

```
int chn_connect (int chan_id, int side);
```

Argument:

- int chan_id: channel identification number from 0 to MAX_NUM_CHAN-1
- int side: one of ‘CHAN_HOST’ or ‘CHAN_TARET’
    - ‘CHAN_HOST’ has responsibility to maintain the channel.

Return:

- 0: on successful completion
- !=0: on failure

Synopsis: It tries to connect the channel with the channel id.

### 2.2.4 closing: chn_close()

Function prototype:

```
int chn_close (int chan_id);
```

Argument:

[2] MAX_NUM_CHAN can be found in ‘cosim_ipc.h’ and 20 by default.

Copyright © 2021 Ando Ki

- int chan_id: channel identification number from 0 to MAX_NUM_CHAN-1

Return:

- 0: on successful completion
- !=0: error code number on failure

Synopsis: It closes the channel specified ‘chnnel_id’ argument. It closes all channels when the argument is ‘-1’.

## 2.3 Sending message

### 2.3.1 blocking: chn_send()

Function prototype:

```
int chn_send (int chan_id, int bnum, void *buf);
```

Argument:

- int chan_id: channel id
- int bnum: number of bytes in the buffer pointed by ‘buf’
- void* buf: pointer to the buffer containing data to send

Return:

- bnum: on successful completion
- -1: error code number on failure; use ‘chn_erro_num()’ to get error code.

Synopsis: It sends ‘bnum’ bytes of data stored in ‘buf[]’ array through the specified channel.

Note: It is blocking version of ‘chn_send_nb()’.

### 2.3.2 non-blocking: chan_send_nb()

Function prototype:

```
int chn_send_nb (int chan_id, int bnum, void *buf);
```

Argument:

- int chan_id: channel id
- int bnum: number of bytes in the buffer pointed by ‘buf’
- void* buf: pointer to the buffer containing data to send

Return:

- >=0: the number of bytes sent successfully
- -1: on failure; use ‘chn_erro_num()’ to get error code.

Copyright © 2021 Ando Ki

Synopsis: It tried to send ‘bnum’ bytes of data stored in ‘buf[]’ array through the channel identified by ‘chan_id’ and returns the number of bytes actually sent.

Note: It is non-blocking version of ‘chn_send()’.

### 2.3.3 core function: _chn_send()

Function prototype:

```
static int  _chn_send (int chan_id, int len, void *buf, int nonblock);
```

Argument:

- int chan_id: channel handler
- int len: number of bytes to send
- void *buf: buffer containing data to send
- int nonblock: 0 means blocking mode, 1 means non-blocking mode

Return:

- >=0: on successful completion; which is the number of bytes sent successfully
- <0: on failure

Synopsis: It sends ‘bnum’ bytes of data stored in ‘buf[]’ array through the specified channel, when ‘nonblock’ is 0. It tried to send ‘bnum’ bytes of data stored in ‘buf[]’ array through the channel identified by ‘chan_id’ and returns the number of bytes actually sent, when ‘nonblock’ is 1.

Note: It is core function of ‘chn_send()’ and ‘chn_send_nb()’.

## 2.4 Receiving message

### 2.4.1 blocking: chn_recv()

Function prototype:

```
int chn_recv (int chan_id, int bnum, void *buf);
```

Argument:

- int chan_id: channel id
- int bnum: number of bytes to be received
- void* buf: pointer to the buffer where received data will be written

Return:

- bnum: on successful completion; which is the number of bytes received successfully
- -1: on failure

Copyright © 2021 Ando Ki

Synopsis: It received up to ‘bnum’ bytes of data and stores them in ‘buf[]’ array through the channel identified by ‘chan_id’.

Note: It is blocking version of ‘chn_recv()’.

### 2.4.2 non-blocking: chn_recv_nb()

Function prototype:

```
int chn_recv_nb (int chan_id, int bnum, void *buf);
```

Argument:

- int chan_id: channel id
- int bnum: maximum number of bytes to be received
- void* buf: pointer to the buffer where received data will be written

Return:

- >0: on successful completion; which is the number of bytes received successfully
- <0: on failure

Synopsis: It tried to receive up to ‘bnum’ bytes of data and stores them in ‘buf[]’ array through the channel identified by ‘chan_id’ and returns the number of bytes actually received.

Note: It is non-blocking version of ‘chn_recv()’.

### 2.4.3 core function: _chn_recv()

Function prototype:

```
static int _chn_recv (int chan_id, int len, char *buf, int nonblock);
```

Argument:

- int chan_id: channel handler
- int len: number of bytes to receive
- char *buf: buffer to store data
- int nonblock: 0 means blocking mode, 1 means non-blocking mode

Return:

- >=0: on successful completion
- <0: error code number on failure

Synopsis: It received up to ‘len’ bytes of data and stores them in ‘buf[]’ array through the channel identified by ‘chan_id’, when ‘nonblock’ is 0. It tried to receive up to ‘len’ bytes of data and stores them in ‘buf[]’ array through the

Copyright © 2021 Ando Ki

channel identified by ‘chan_id’ and returns the number of bytes actually received, when ‘nonblock’ is 1.

Note: It is core function of ‘chn_recv()’ and ‘chn_recv_nb()’.

## 2.5 additional API

### 2.5.1 handle: chn_handle()

Function prototype:

```
void *chn_handle (int chan_id, int *type);
```

Argument:

- ✧ int chan_id: channel id
- ✧ int *type: type of channel; ‘CHAN_HOST(1L)’ for host side, ‘CHAN_TARGET (2L)’ for target side.

Return:

- ✧ channel handler on successful.
- ✧ -1L: on failure

Synopsis: It returns channel handler for the given channel identification.

### 2.5.2 barrier: chn_barrier()

Function prototype:

```
int chn_barrier (int chan_id);
```

Argument:

- ✧ int chan_id: channel id

Return:

- ✧ 0: on successful completion
- ✧ -1: on failure

Synopsis: It tries to synchronizes between host and target by exchanging process identification number.

Note: It should be called after ‘chn_create_conect()’.

### 2.5.3 set verbose level: chn_set_verbose()

Function prototype:

```
void chn_set_verbose (int level);
```

Copyright © 2021 Ando Ki

Argument:

- ✧ int level: verbose level (0 means silence).

Synopsis: It sets verbose level, which determines level of details for message.

### 2.5.4 get verbose level: chn_get_verbose()

Function prototype:

```
int chn_get_verbose (void);
```

Return: verbose level

Synopsis: It returns verbose level.

### 2.5.5 get error number: chn_error_num()

Function prototype:

```
int chn_error_num (void);
```

Return: error message identification number. 0 means no error.

Synopsis: It returns value of ‘error_msg_num’, which indicates execution status.

### 2.5.6 get error message: chn_error_msg()

Function prototype:

```
char* chn_error_msg(int errn) ;
```

Return: pointer to error message string.

Synopsis: It returns pointer to error message string.

# 3 Typical usage

## 3.1 Blocking case

| Host side | Target side |
|---|---|
| int host(int cid) {<br>    chn_init();<br>    if (chn_create_connect(cid, CHAN_HOST)<0)<br>                    return 1;<br>    printf("sender established channel\n");<br>   chn_barrier(cid); | int target(int cid) {<br>    chn_init();<br>    if (chn_create_connect(cid, CHAN_TARGET)<0)<br>                    return 1;<br>   printf("receiver established channel\n");<br>   chn_barrier(cid); |

Copyright © 2021 Ando Ki

| | |
|---|---|
| sprintf(buf, "It is sender at HOST.");<br>chn_send(cid, MLENG, buf);<br>chn_recv(cid, MLENG, buf);<br>printf("received \"%s\"\n", buf);<br><br>if (chn_close(cid)<0) {<br>printf("fail to close channel\n");<br>return 1;<br>}<br>} | chn_recv(cid, MLENG, buf);<br>printf("received \"%s\"\n", buf);<br>sprintf(buf, "It is receiver at TARGET.");<br>chn_send(cid, buf, MLENG);<br><br>if (chn_close(cid)<0) {<br>printf("fail to close channel\n");<br>return 1;<br>}<br>} |

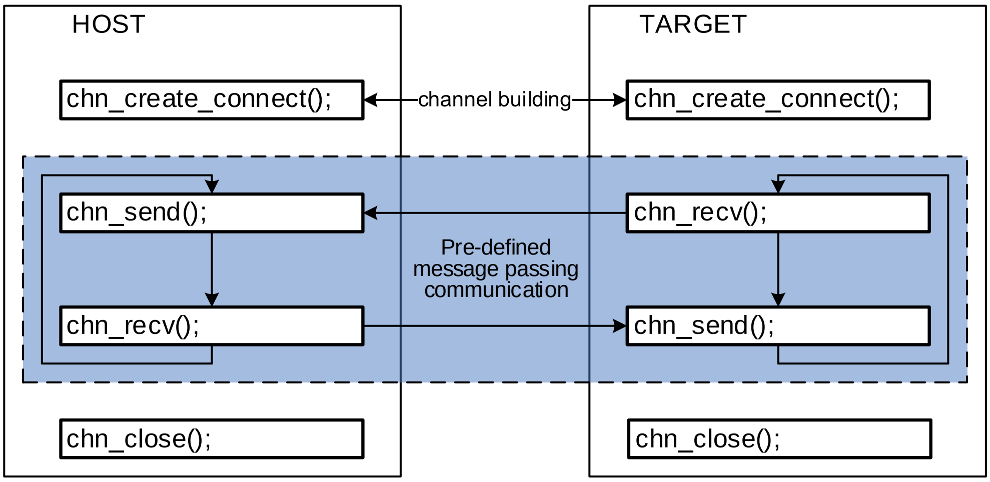

**Figure 2: IPC flow**

## 3.2 Non-blocking case

| Host side | Target side |
|---|---|
| int host(int cid) {<br>chn_init();<br>if (chn_create_connect(cid, CHAN_HOST)<0)<br>return 1;<br>printf("sender established channel\n");<br>chn_barrier(cid);<br><br>sprintf(buf, "It is sender at HOST: %d.", i);<br>int len = strlen(buf) + 1;<br>int sent = 0;<br>do { sent = chn_send_nb(cid, len, buf);<br>if (sent<0) {<br>printf("Unsuccessful sending.\n");<br>break;<br>}<br>len = len - sent;<br>} while (len>0);<br><br>if (chn_close(cid)<0) {<br>printf("fail to close channel\n");<br>return 1; | int target(int cid) {<br>chn_init();<br>if (chn_create_connect(cid, CHAN_TARGET)<0)<br>return 1;<br>printf("receiver established channel\n");<br>chn_barrier(cid);<br><br>int recv = 0;<br>do { recv = chn_recv_nb(cid, MLENG, buf);<br>if (recv<0) {<br>printf("Unsuccessful receiving.\n");<br>break;<br>}<br>} while (recv==0);<br>printf("received \"%s\"\n", buf);<br><br>if (chn_close(cid)<0) {<br>printf("fail to close channel\n");<br>return 1;<br>}<br>} |

Copyright © 2021 Ando Ki

| | |
|---|---|
| }<br>} | |

## 4 Tips

- Use ‘ipcs’ command to obtain the status of all System V IPC objects.
    - use ‘$ipcs -q’
- Use ‘ipcrm’ command to remove the specified System V IPC objects.
    - use ‘$ipcrm -q num’

## 5 Wish list

- Closing channel by any side of channel using signal (i.e., kill()).
- Extend this idea to socket-based IPC.

## 6 References

[1] Ando Ki, Co-Simulation Library for AMBA AXI BFM using DPI/VPI, Aug. 20, 2021.

## 7 Revision history

- 2021.08.01: V0R1 released by Ando Ki.
- 2021.07.01: Started by Ando Ki (andoki@gmail.com)

-- End of Document --

Copyright © 2021 Ando Ki