IC-Service IPC Detailed design Document

Ver.0.3.2

2021/06/29

＜Table of Contents＞

1. Document Purpose

In this document, we design the program configuration, communication protocol, and data related to IPC between IC-Service and Cluster.

2. Change history

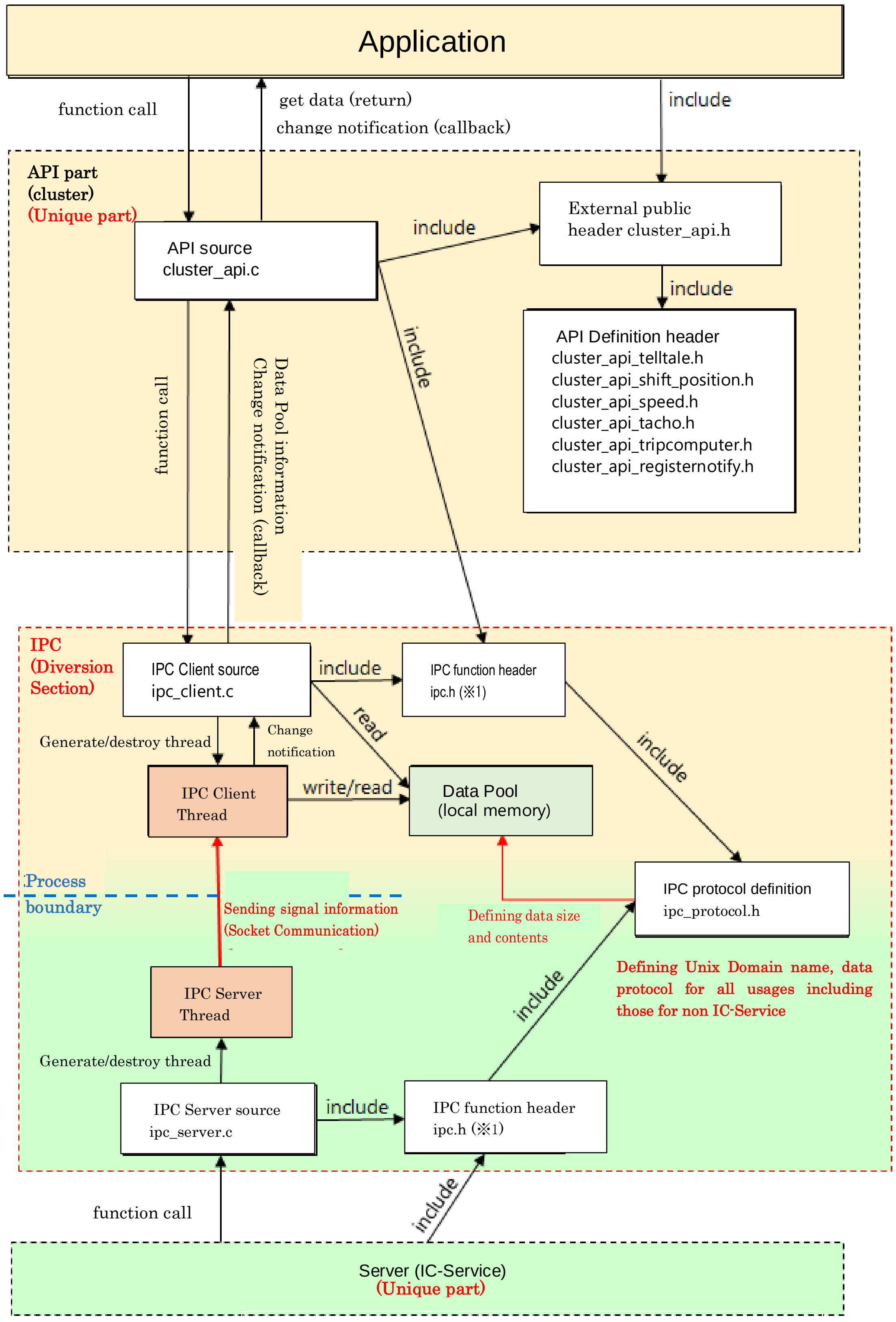
Version	Date	Detail	Editor
0.1.0	2020/11/26	・ 初回作成(ラフ)	
0.2.0	2020/12/11	・ ひと通り設計	
0.3.0	2020/12/22	・ IPC 流用可能部を実際に実装してから整理	
0.3.1	2021/06/21	・ 英語の翻訳	
0.3.2	2021/06/29	・ 図は英語で翻訳	
		・	
		・	
		・	
		・	

3. Detailed design of Program configuration

3.1. Library program configuration table

Configuration items	Overview
アプリ Application	Cluster UI part. - Call API functions for Cluster.
Cluster	IC-Service data acquisition API Part. - Provide the signal information obtained from IC-Service to the application. - Operate as part of the application process (Layer part where functions are called from the application)
IC-Service	The part has the internal information of the car as signal information. - Send signal information to the Cluster. - Operate as a separate process from the application process.
IPC	Inter Process Communication - Perform the function of Inter Process Communication which sends signal information from IC-Service to Cluster. - This part is designed and implemented to be used for other than communication between Cluster and IC-Service.

3.2. Library program configuration diagram



(※1) In fig, ipc.h described two times in structure, but both are the same

3.3. IPC Diversion Section Design

In 3.2, the IPC part is described for communication between Cluster and IC-Service. The IPC part is designed and implemented even for a communication protocol between other modules.

3.3.1. IPC Diversion Section Design overview

Design Considering the diversion part of the IPC as follows:

- Prerequisites
  - Communication shall be made using Unix Domain Socket.
  - *epoll* shall be used to monitor file descriptors
  - The implementations for both Client and Server are combined into one .so file.
  - One external public header *Include* from both Client and Server.
- External public header file specifications
  - The header file name is *ipc.h*, shown in 3.2.
  - *Include* the header from both the Client side and the Server side.
  - *ipc.h* has functions and macros to provide each mechanism of IPC for Client and Server.
  - For all usage in *ipc_protocol.h* (from *include ipc.h*), "data protocol" and "Unix Domain name" are defined one by one.
  - Select one Client and Server from usage defined in *ipc_protocol.h*.
- Communication and data specifications
  - A thread for IPC communication is generated in the IPC diversion part, client and a server communicate on the thread.
    - One process contains a single thread. The same applies when there are multiple domains used.
    - When the Client-side thread starts, it allocates memory for the Data Pool.
  - The thread on the Client-side receives the data sent from the server and writes the received data to the Data Pool.
    - Sending and receiving all structures defined as the data protocol for the Domain.
    - At this time, the Client-side also detects a change in the Data Pool, Callback notification of the change to the Client-side application.
- Limitations
  - The communication direction is only from Server to Client.
  - Prohibit multiple servers from using the same Domain.

3.3.2. IPC Diversion Section　 Use Case

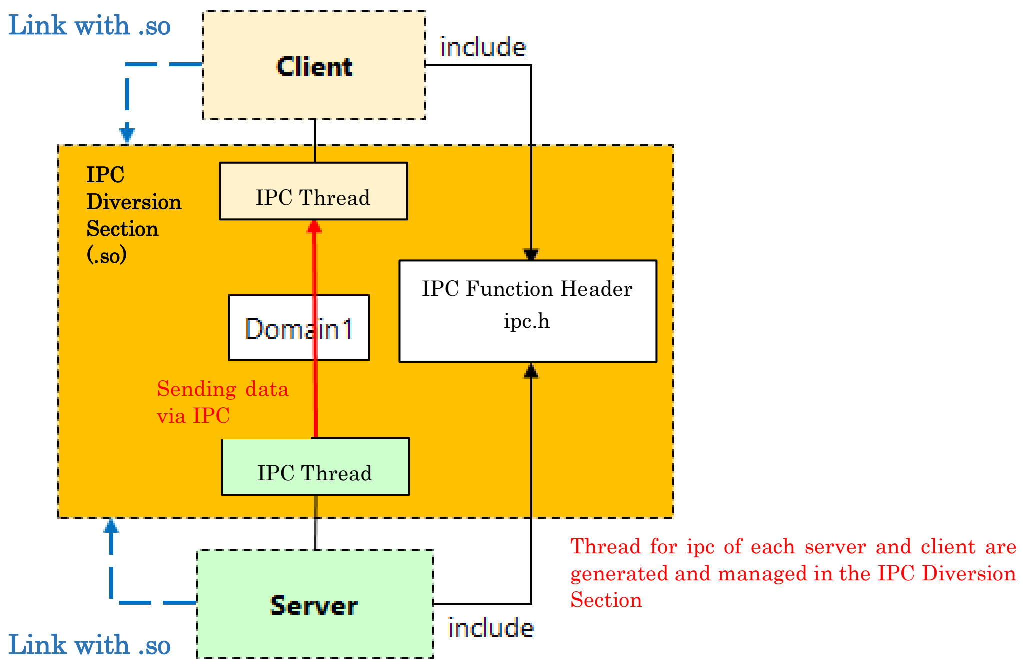

Link with .so
Client
include
IPC Diversion Section (.so)
IPC Thread
IPC Function Header ipc.h
Domain1
Sending data via IPC
IPC Thread
Thread for ipc of each server and client are generated and managed in the IPC Diversion Section
Server
include
Link with .so

Fig 3.3.2-1　One-to-one (Example)

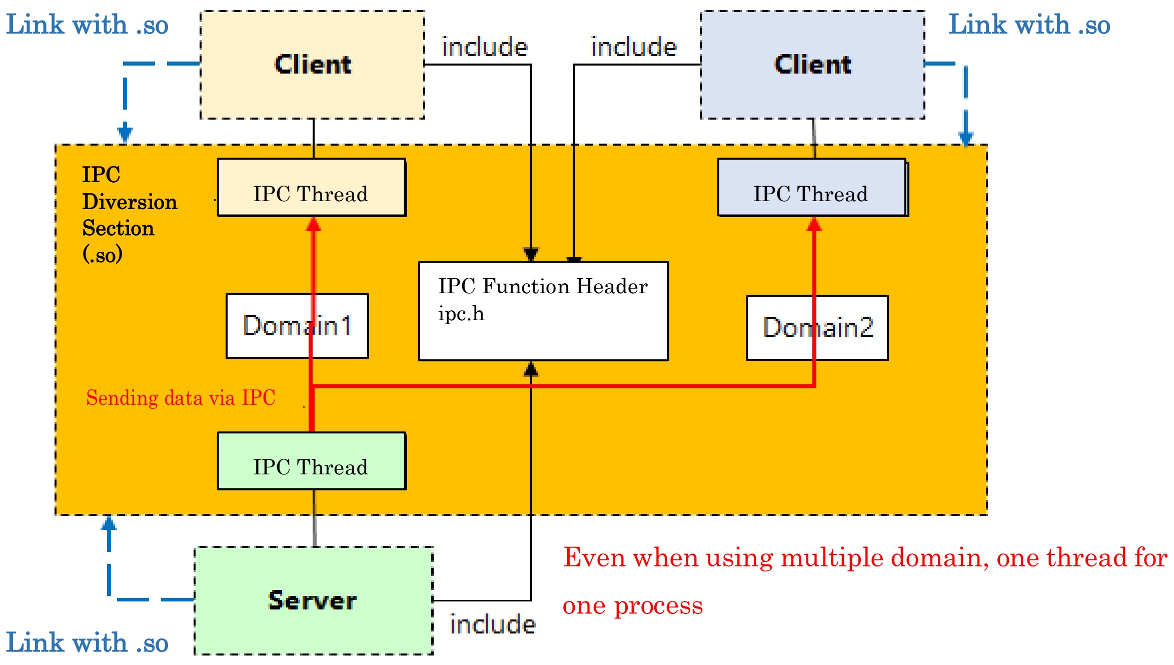


Fig 3.3.2-2　Multiple Domain use (Example)

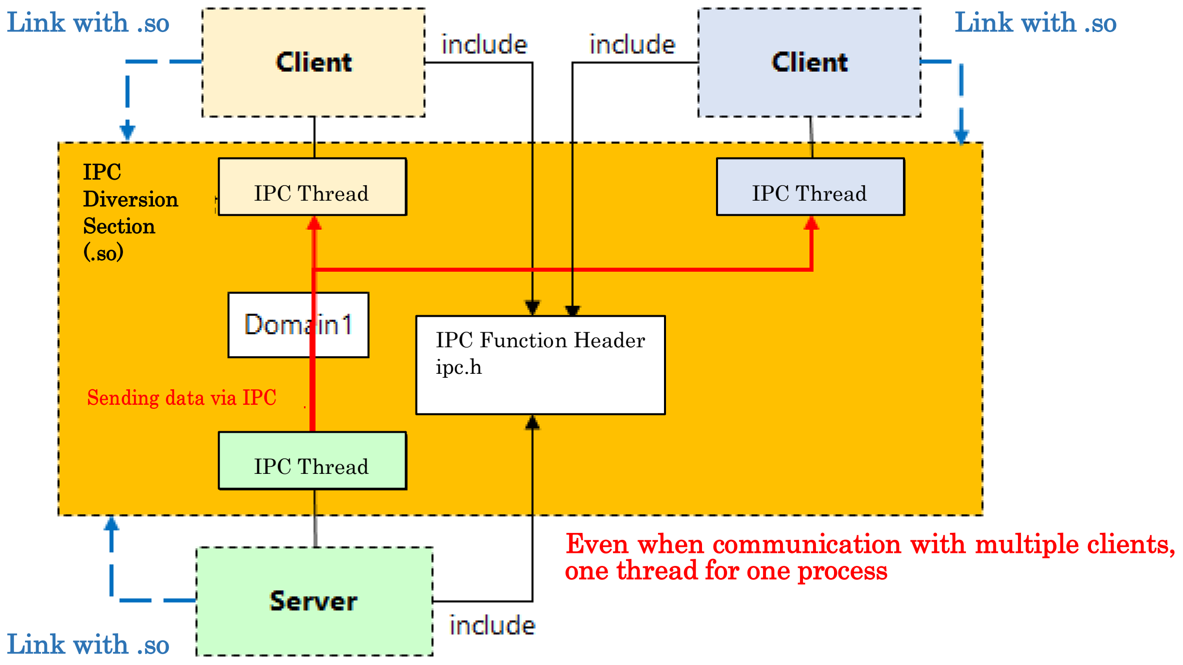


Fig 3.3.2-3　One Domain, Multiple Clients (Example)

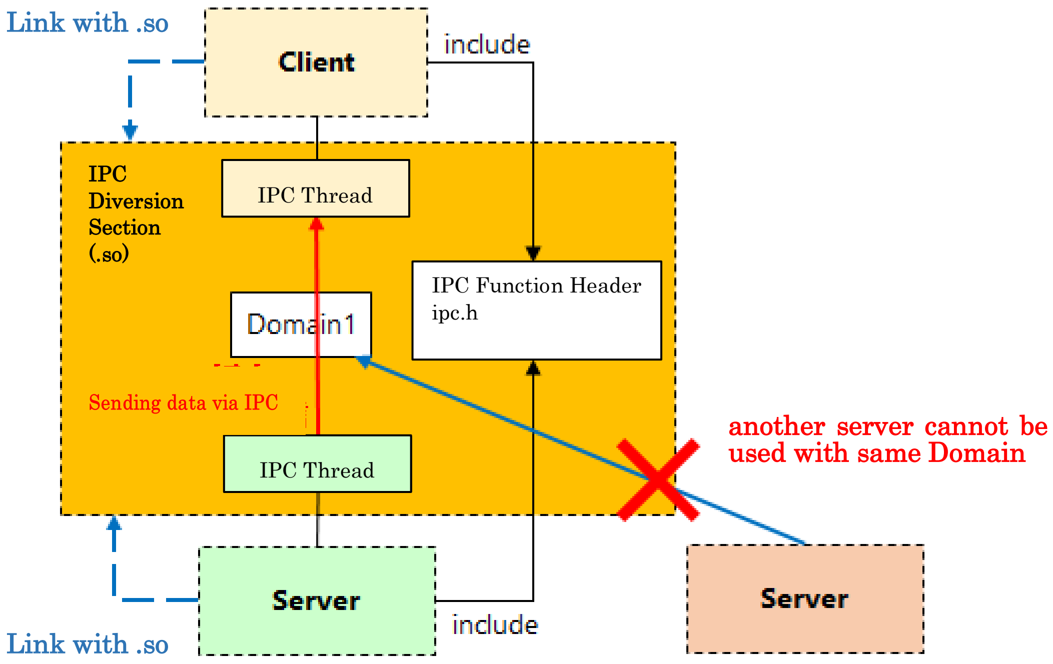


Fig 3.3.2-4　Unavailable Domain Use (Example)

3.3.3. IPC Diversion Section Mechanism Providing Function Design

Preparing some functions for Client and Server that use IPC diversion part.
These functions are declared in *ipc.h* and provided for Client and Server.

Function name	Function Overview	User
ipcServerStart()	Start IPC Server	Server
ipcSendMessage()	Send data to Client.	Server
ipcServerStop()	Stop IPC Server.	Server
ipcClientStart()	Start the IPC Client and connect to the IPC Server in the same domain.	Client
ipcReadDataPool()	Read all the data received from the Server which stored at Data Pool.	Client
ipcRegisterCallback()	Register the callback function to receive the Data Pool change notifications.	Client
ipcClientStop()	Stop IPC Client.	Client

Next section, Describing functions usage way (idea).

3.3.3.1. Starting IPC for Server and Client

ipcServerStart() and ipcClientStart() are called by specifying the usage type (enum) as an argument.
Call for the IC-Service as follows:

- Server-side (called before Client-side)
  ipcServerStart(IPC_USAGE_TYPE_IC_SERVICE);
  - Execute socket(), bind(), listen() as a servers for IC-Service usage.
    Wait for a connection from a Client that specifies the same usage type.
  - Create a thread for monitoring the connection status from the Client.
    - Monitor state by epoll, connect to Client by accept(),disconnect from Client by close().

- Client-side
  ipcClientStart(IPC_USAGE_TYPE_IC_SERVICE);

- ■ Create a Data Pool area for storing received data.
  Data Pool size depends on the type of usage.
- ■ Execute socket() and connect() as a client for IC-Service usage.
  Connect to a server with the same usage type

3.3.3.2. Client side Setting of Callback function for change receiving

ipcRegisterCallback() is called by specifying the usage type (enum) and the callback function.
Call only from the Client side.
The callback function specified here is used for notification of changes in data received from the Server.
For IC-Service is executed as follows (Example):

- ipcRegisterCallback(IPC_USAGE_TYPE_IC_SERVICE, changeNotifyCb)
  - ➢ Able to receive callback notifications for all data monitored in the usage.
  - ➢ In IC-Service, only the change of the parameter corresponding to 3-2-1 TellTale in the attachment「IC-Service_API_rev0.4.docx」 is monitored.
    (For example, if the value corresponding to getSeatbelt() of Telltale changes, it will be notified, but if the value corresponding to getFrontRightSeatbelt() does not correspond to 3 -2 -1 TellTale, notification will not be sent).
- Store only one callback function for each usage type. If ipcRegisterCallback() is called again for the same usage type, it will be overwritten.
- See 3.3.5 for callback specifications.

3.3.3.3. Sending from Server side

ipcSendMessage() is called by specifying the usage type (enum) and sending data (address, size).
Call only from Server.
For IC-Service execute as follows (Example)

- ipcSendMessage(IPC_USAGE_TYPE_IC_SERVICE, &dataIcService, sizeof(dataIcService));
  - ■ In the 2nd and 3rd arguments, specify a head address and the size of sending data.
  - ■ In the case of IC-Service, the IPC_DATA_IC_SERVICE_S structure is used as the sending data.
  - ■ When executed, data is sent to the Client (all if there are multiple) that specify the same usage type.

3.3.3.4. Reading received data from Server

ipcReadDataPool() is called by specifying the usage type (enum), the received data storage address, and the received data storage size.

Call only from Client.

For IC-Service execute as follows (Example):

- ipcReadDataPool(IPC_USAGE_TYPE_IC_SERVICE, &dataIcService, &size);
  - Data received from the Server in the Data Pool.
  - When calling this function, read the entire contents of the Client DataPool.
  - In the second argument, specify the address to store the contents of DataPool. Prepare the entity of IPC_DATA_IC_SERVICE_S structure for IC-Service.
  - The third argument is input/output, at input specifying the size of the storage destination specified by the second argument, and at the output substitute real stored size.

3.3.3.5. Client and Server Termination

ipcServerStop() and ipcClientStop() are called by specifying the usage type (enum) argument. For IC-Service is called as follows.

- Client side (recommended to call before Server)
  - ipcClientStop(IPC_USAGE_TYPE_IC_SERVICE);
    - Disconnect from a Server (shutdown(), close())
    - Releasing the Data Pool area.
- Server side
  - ipcServerStop(IPC_USAGE_TYPE_IC_SERVICE);
    - Disconnect from all connected clients (close())
    - Close of own Socket.

3.3.4. Variable environment for Setting Socket File Path

When ipcServerStart() is executed, generate a Socket file for communication between Server and Client (using Unix Domain Socket).

By default, a Socket file is created at the server application execution location, but the environment variable **IPC_DOMAIN_PATH** can specify the Socket file generation path.

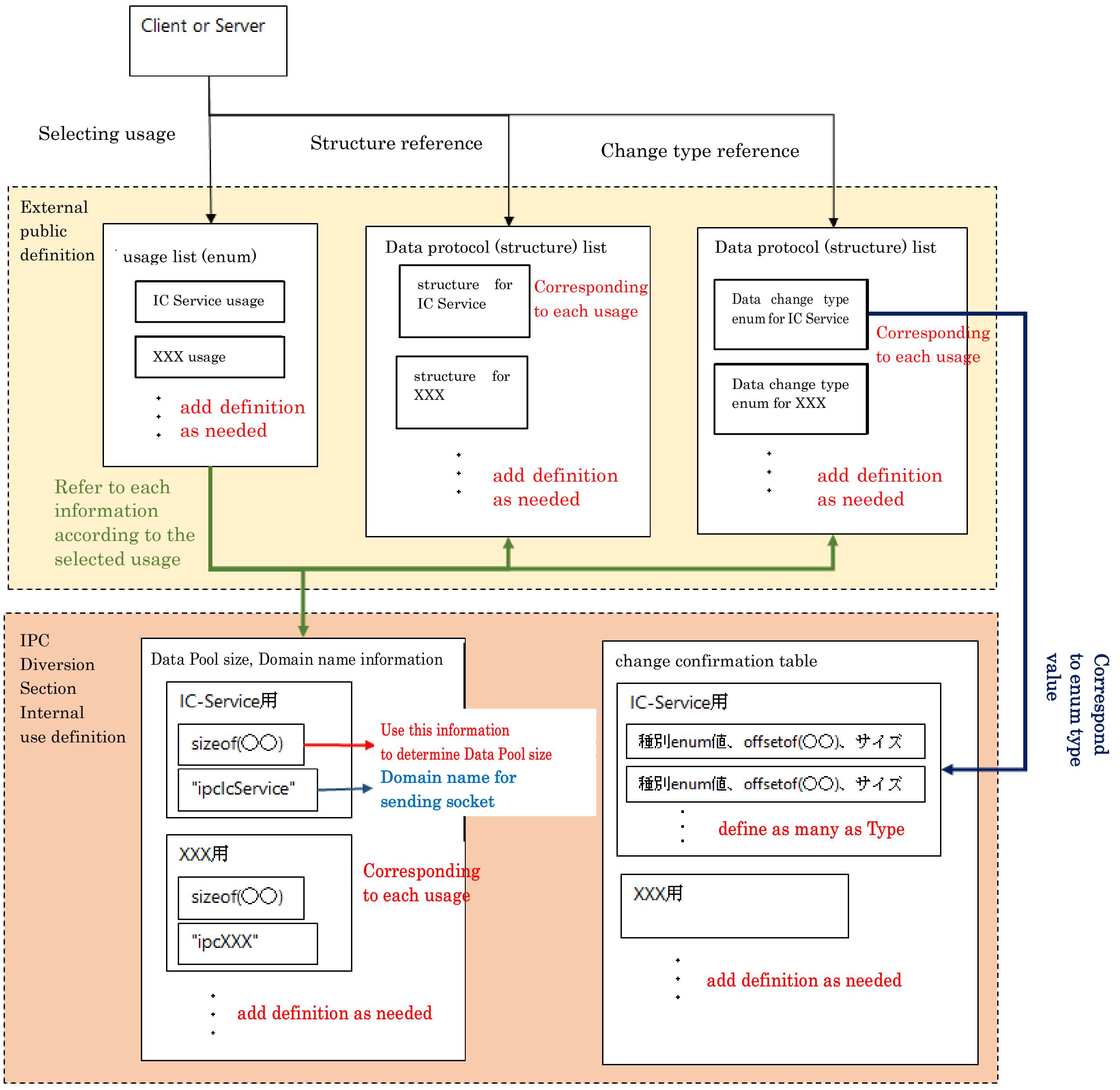
For example, if **IPC_DOMAIN_PATH** is set to "/tmp", then Socket files will be generated for IC-Service with the path name "/tmp/ipcIcService".

3.3.5. Definition method of Data protocol

Data sent and received for each usage, and the Unix Domain name for communication are defined within ipc_protocol.h.

The definition is as follows.

- An enum definition for Client/Server selects a usage type.
- Data protocol (structure) definition corresponding to the usage type (referable from Client/Server)
- enum definition for changed data type corresponding to usage type (referable from Client / Server)
- Data Pool size and Domain name definition corresponding to usage type (used inside IPC).
- Change detection address (offset) table corresponding to the usage type (used inside IPC).


Client or Server
Selecting usage
Structure reference
Change type reference
External public definition
usage list (enum)
IC Service usage
XXX usage
add definition as needed
Data protocol (structure) list
structure for IC Service
Corresponding to each usage
structure for XXX
add definition as needed
Data protocol (structure) list
Data change type enum for IC Service
Corresponding to each usage
Data change type enum for XXX
add definition as needed
Refer to each information according to the selected usage
IPC Diversion Section Internal use definition
Data Pool size, Domain name information
IC-Service用
sizeof(〇〇)
"ipcIcService"
Use this information to determine Data Pool size
Domain name for sending socket
XXX用
sizeof(〇〇)
"ipcXXX"
Corresponding to each usage
add definition as needed
change confirmation table
IC-Service用
種別enum値、offsetof(〇〇)、サイズ
種別enum値、offsetof(〇〇)、サイズ
define as many as Type
XXX用
add definition as needed
Correspond to enum type value

3.3.6. Detecting Data change and specification of notification callback

When data is sent from the Server to the Client, the Client-side detects changes in the data and calls the callback function for the change notification.

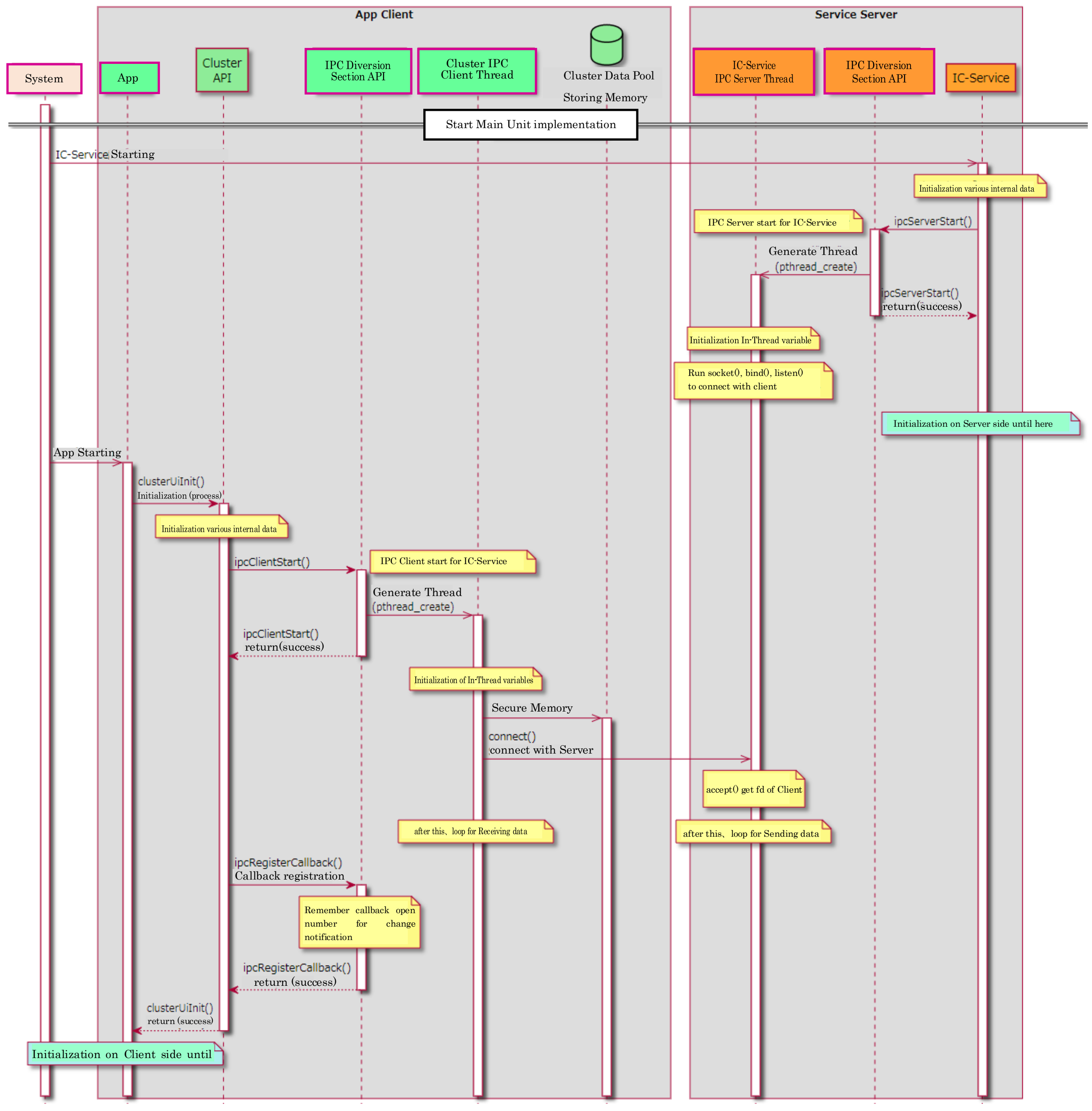
- Change detection method (Client-side, IPC thread processing)
  - For each usage, prepare a "change confirmation table" which arranges the offset address and size within the structure related to detected data for change.
    (all structures are processed in the same way)
  - When data sent from the Server to the Client, the Client temporarily stores the received data in a local variable within the thread.
  - Regarding local variables and Data Pool, compare each data according to the "change confirmation table".
  - When a changed value is detected, the Client notified a callback about that value.

- Callback specification
  - Ask the Client to register a callback function in advance.
  - The callback function definition is as follows (It can handle any usage);
    typedef void (*IPC_CHANGE_NOTIFY_CB)(void * pData, signed long size, int kind);
    ・pData : Store address of changed data
    ・size： The size of the data
    ・kind： Data type (enum value according to usage)

- Regarding creating a change confirmation table
  - The offset table of each data in the structure can be easily created using offsetof().

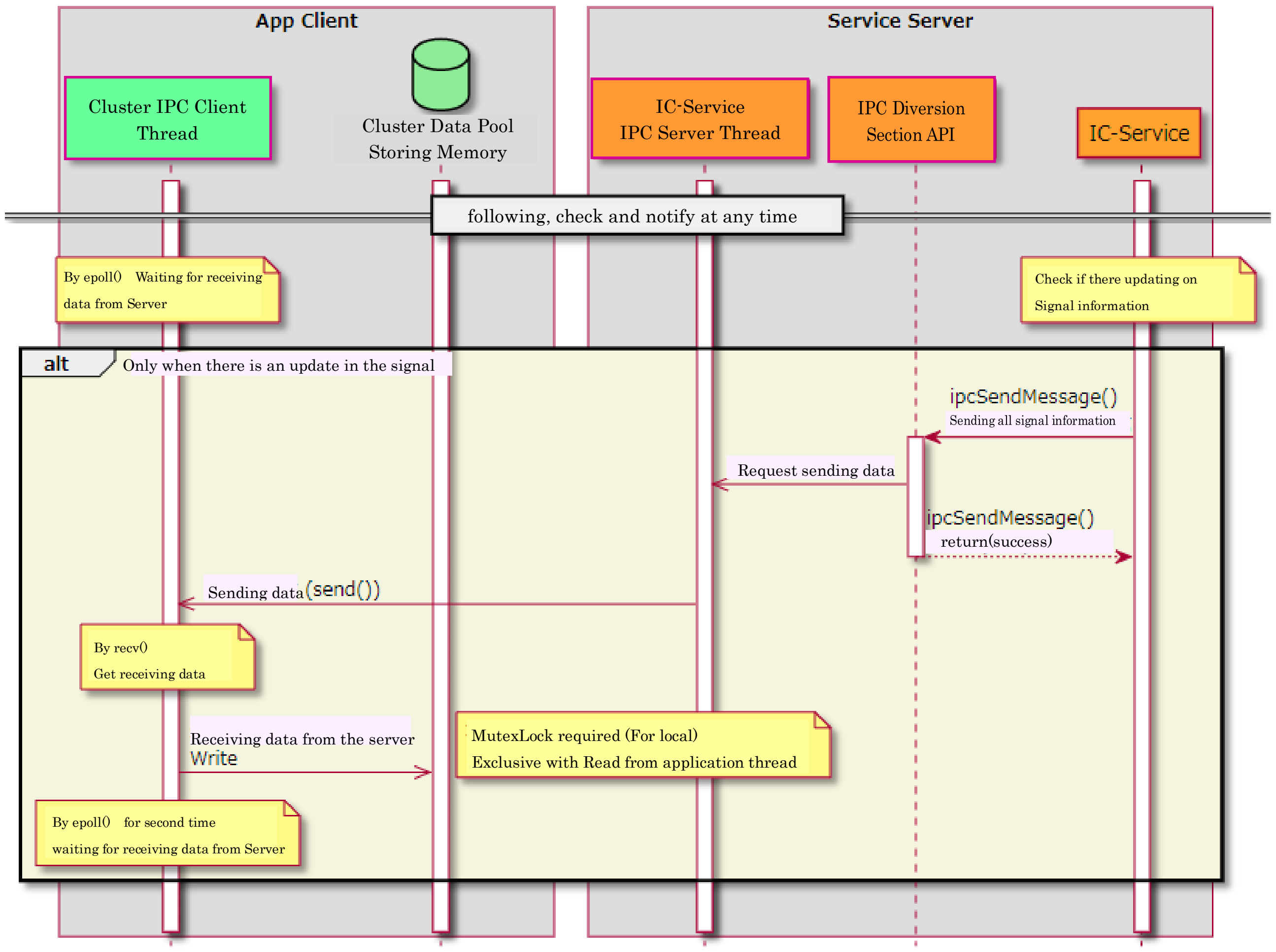
3.4. Library Detail Sequence

3.4.1. Startup/Initialization

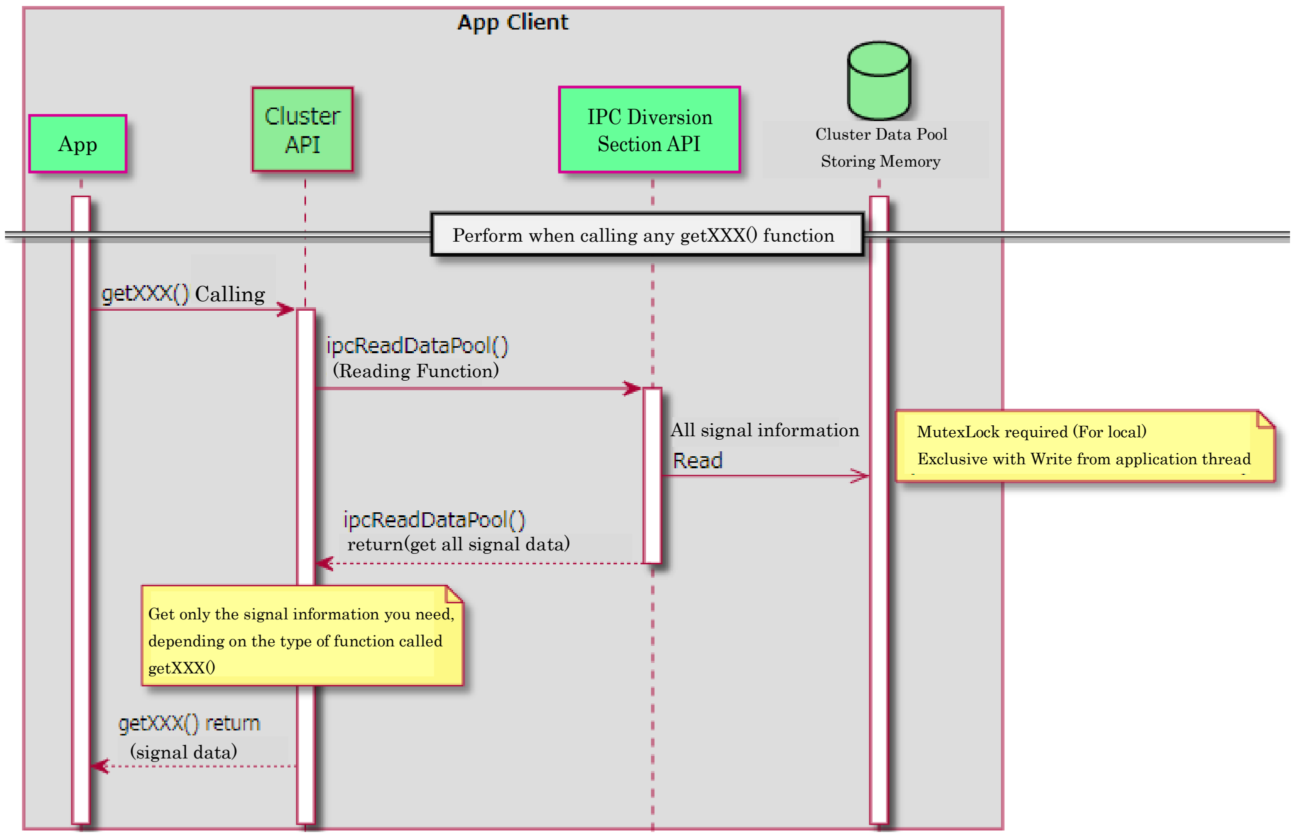


App Client
Service Server
System
App
Cluster API
IPC Diversion Section API
Cluster IPC Client Thread
Cluster Data Pool
Storing Memory
IC-Service IPC Server Thread
IPC Diversion Section API
IC-Service
Start Main Unit implementation
IC-Service Starting
Initialization various internal data
IPC Server start for IC-Service
ipcServerStart()
Generate Thread (pthread_create)
ipcServerStart() return(success)
Initialization In-Thread variable
Run socket(), bind(), listen() to connect with client
Initialization on Server side until here
App Starting
clusterUiInit() Initialization (process)
Initialization various internal data
ipcClientStart()
IPC Client start for IC-Service
Generate Thread (pthread_create)
ipcClientStart() return(success)
Initialization of In-Thread variables
Secure Memory
connect() connect with Server
accept() get fd of Client
after this, loop for Receiving data
after this, loop for Sending data
ipcRegisterCallback() Callback registration
Remember callback open number for change notification
ipcRegisterCallback() return (success)
clusterUiInit() return (success)
Initialization on Client side until

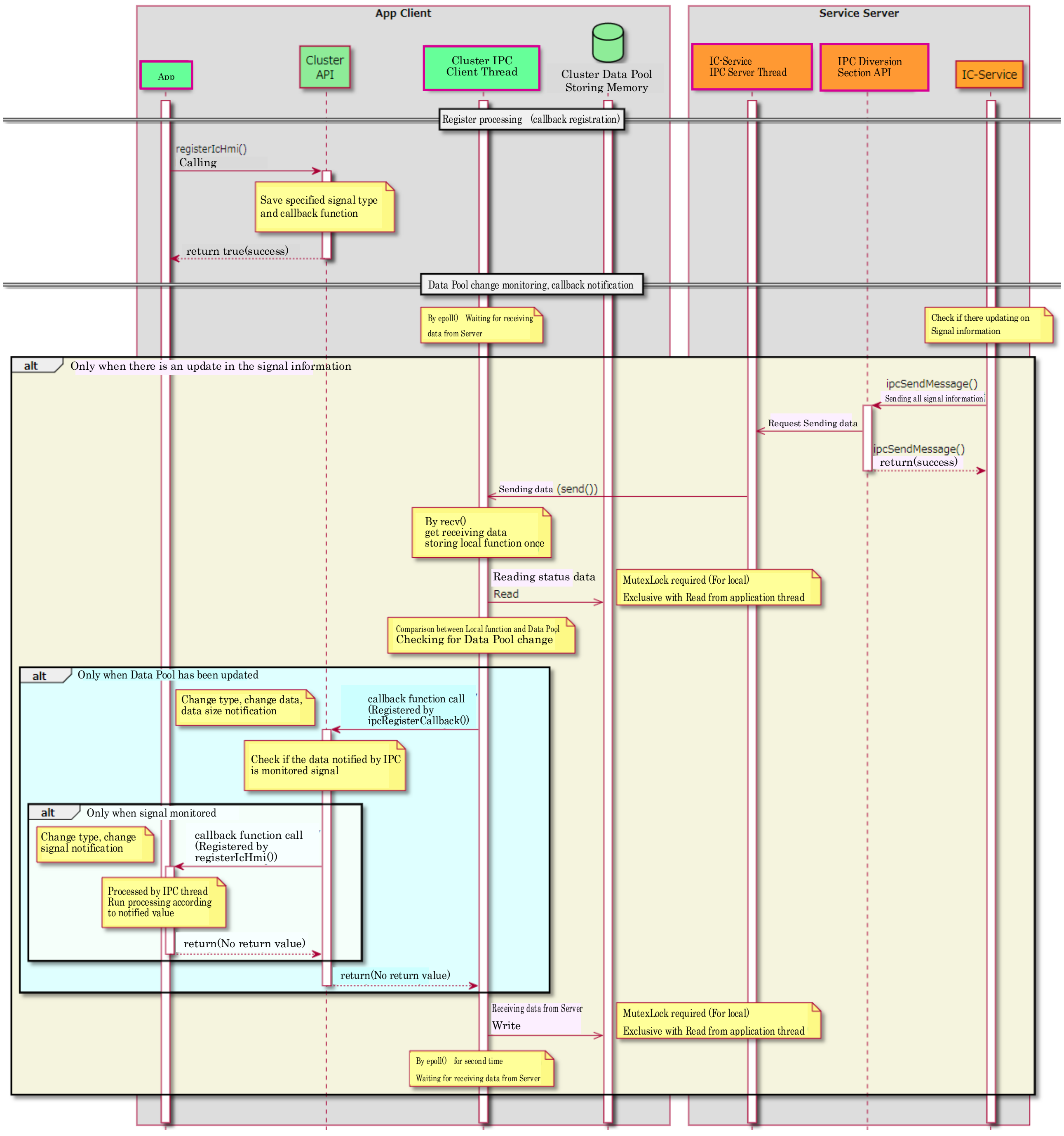
3.4.2. Update Data Pool information


App Client
Service Server
Cluster IPC Client Thread
Cluster Data Pool Storing Memory
IC-Service IPC Server Thread
IPC Diversion Section API
IC-Service
following, check and notify at any time
By epoll() Waiting for receiving data from Server
Check if there updating on Signal information
alt
Only when there is an update in the signal
ipcSendMessage()
Sending all signal information
Request sending data
ipcSendMessage()
return(success)
Sending data (send())
By recv() Get receiving data
Receiving data from the server Write
MutexLock required (For local) Exclusive with Read from application thread
By epoll() for second time waiting for receiving data from Server

3.4.3. Acquisition API Process


App Client
App
Cluster API
IPC Diversion Section API
Cluster Data Pool Storing Memory
Perform when calling any getXXX() function
getXXX() Calling
ipcReadDataPool() (Reading Function)
All signal information Read
MutexLock required (For local)
Exclusive with Write from application thread
ipcReadDataPool() return(get all signal data)
Get only the signal information you need, depending on the type of function called getXXX()
getXXX() return (signal data)

3.4.4. Register/Notify Process

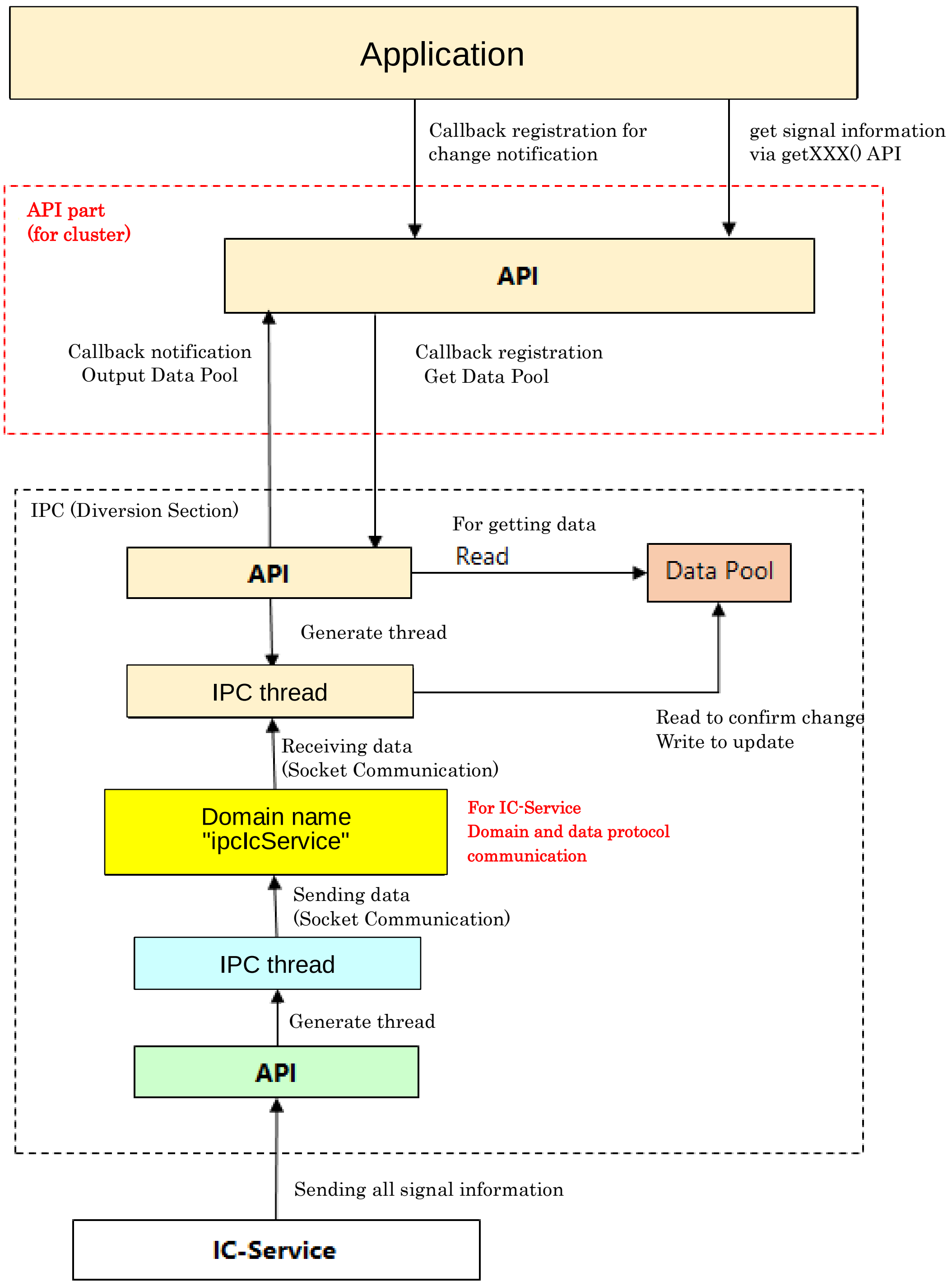

App Client
Service Server
App
Cluster API
Cluster IPC Client Thread
Cluster Data Pool Storing Memory
IC-Service IPC Server Thread
IPC Diversion Section API
IC-Service
Register processing (callback registration)
registerIcHmi() Calling
Save specified signal type and callback function
return true(success)
Data Pool change monitoring, callback notification
By epoll() Waiting for receiving data from Server
Check if there updating on Signal information
alt
Only when there is an update in the signal information
ipcSendMessage()
Sending all signal information
Request Sending data
ipcSendMessage() return(success)
Sending data (send())
By recv() get receiving data storing local function once
Reading status data
Read
MutexLock required (For local) Exclusive with Read from application thread
Comparison between Local function and Data Pool
Checking for Data Pool change
alt
Only when Data Pool has been updated
Change type, change data, data size notification
callback function call (Registered by ipcRegisterCallback())
Check if the data notified by IPC is monitored signal
alt
Only when signal monitored
Change type, change signal notification
callback function call (Registered by registerIcHmi())
Processed by IPC thread Run processing according to notified value
return(No return value)
return(No return value)
Receiving data from Server
Write
MutexLock required (For local) Exclusive with Read from application thread
By epoll() for second time Waiting for receiving data from Server

4. Communication protocol

4.1. Communication protocol Design

Design data to communicate between Cluster and IC-Service.

4.1.1. Data & Communication Configuration


Application
Callback registration for change notification
get signal information via getXXX() API
API part (for cluster)
API
Callback notification Output Data Pool
Callback registration Get Data Pool
IPC (Diversion Section)
For getting data
Read
Data Pool
API
Generate thread
IPC thread
Read to confirm change Write to update
Receiving data (Socket Communication)
Domain name "ipcIcService"
For IC-Service Domain and data protocol communication
Sending data (Socket Communication)
IPC thread
Generate thread
API
Sending all signal information
IC-Service

4.1.2. Data protocol

- The Domain name for IC-Service used in IPC is defined as "ipcIcService" in ipc_protocol.h
- IC-Service is Server, and Cluster is Client. All signal information is sent from IC-Service to Cluster (application side).
- All signal information shall be combined into a single structure, sent and received by the IPC diversion section.
  - This structure is defined in ipc_protocol.h for IC-Service.
  - In " Data Pool サイズ整理.xlsx" attachment, assumed that all the data collected into a structure.
    (Based on the return value of each API in IC-Service_API_rev0.4.docx)
  - Every time would send all signal information together, even if only some signals changed
- The sending time from IC-Service to Cluster is assumed to be about 10 msec.
- The getXXX () function call from the application returns the contents stored currently in the Data Pool.
  - Do not return all signals; output only the signal information corresponding to getXXX() to the application as the function's return value.
- The acquisition signal and callback function registered by registerIcHmi() from the application are stored and managed in Cluster.
- According to the notifyIcHmi() specification, the callback function sends only the changed signal information to the application.
- When a change notification callback is received from the IPC diversion part, the callback is notified to the application only when the signal is changed by registerIcHmi().

4.2. Data design

4.2.1. Data Pool design for storing all signal information

- All signal information sent from the IC-Service is stored in the Data Pool as described in 4.1.2.
- The size required as a Data Pool for the Domain name "ipcIcService" is 276 bytes in a 32-bit environment and 296 bytes in a 64-bit environment, as described in the Attachment " Data Pool サイズ整理.xlsx ".
- In order to confirm changes in signal information, all Data Pool and comparison signal information will be local within the IPC thread (The size is equal to 276 or 296 bytes.).