InHand OpenDevice Platform Development Guide

Introduction

InHand OpenDevice platform provides various communication protocols which support such as publish/subscribe model, request/response model and pipeline model by utilizing asynchronous message queue. Based on the programming model of InHand OpenDevice platform, customized application not only run more efficiently compared to synchronous programming model, but also with OpenDevice encapsulations communication interface, developer just concentrate on realizing service development with no need to consider low-layer communication problems, which will simplify application development and improve development efficiency.

Asynchronous Clock Programming Model

Asynchronous clock programming model provides clock event management for customized applications. E.g., the information of cellular module is displayed every 30s.
Example code

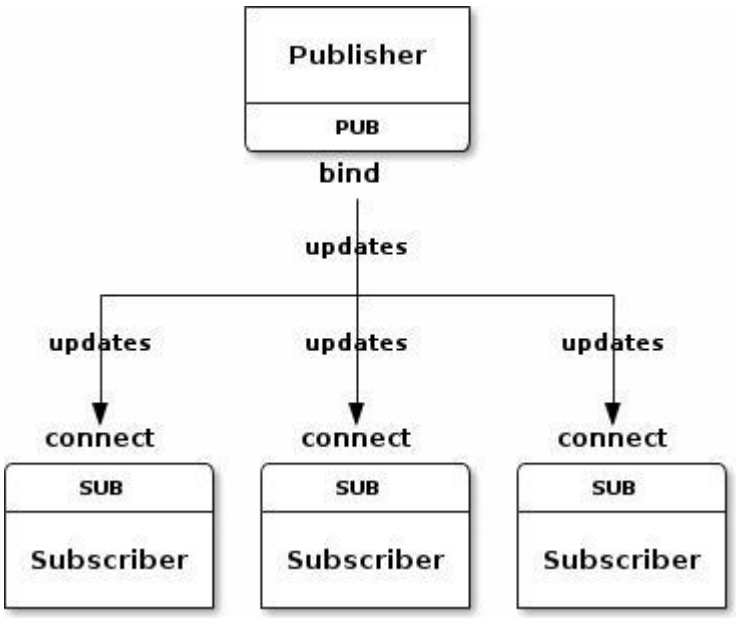
```
####################
#test-timer.py
#show cellular module per 30s
####################
import MessageChannel
import RouterInfo

def _cellinfo_handle(fd, event, obj):
#show cellular info
    MessageChannel.logging.info(RouterInfo.get_cellular_info())
#timeout after 30s
    MessageChannel.startTimer(cellevt, 30)

if __name__ == "__main__":
#open log at level debug
    MessageChannel.logconfig("debug")
#register cellular handle that show cellular info
    cellevt = MessageChannel.registerTimer( _cellinfo_handle)
#start timer
    MessageChannel.startTimer(cellevt, 1)
#register INT signal
    MessageChannel.registerSigInt()
#main loop
    MessageChannel.run()
```

Asynchronous Communication Programming Model

OpenDevice asynchronous communication programming model is compose of Publish/Subscribe programming model, Request/Response programming model and pipeline programming model, which is implement by MessageChannel module, base on nanomsg and libevent. MessageChannel is python class, which generate several subclass, such as MessagePublish, MessageRequest and so on. Customized application not only instantiate those class, but also inherit or overload those class to realize special requirement.

Publish/Subscribe Model

This communication model is used for information publishing or event notification. In the real life, weather forecast notification or discount information publishing is belonged to this type of communication model. The communication diagram of publish and subscribe communication model is shown below.



Notes：

Subscriber cannot receive the messages published before subscription.

There is an example that message publisher publishes the message “Hello World” every 10s.

Example code:

```
########################################
#pubtest.py------publisher test case
#publish 'Hello World' per 10s
########################################
import MessageChannel

def _pubTimeoutHandle(fd, event, obj):
    pubinfo = "Hello world"
        pub.write(pubinfo)
    MessageChannel.logging.info("publish info: %s" %pubinfo)
    MessageChannel.startTimer(pubevt, 10)

if __name__ == '__main__':

    MessageChannel.logconfig('debug')
    #create publisher with ipc address "ipc:///tmp/test-pubsub.ipc"
    pub = MessageChannel.MessagePublish('ipc:///tmp/test-pubsub.ipc')
    pubevt = MessageChannel.registerTimer( _pubTimeoutHandle)
    MessageChannel.startTimer(pubevt, 1)

    MessageChannel.registerSigInt()
    MessageChannel.run()
```

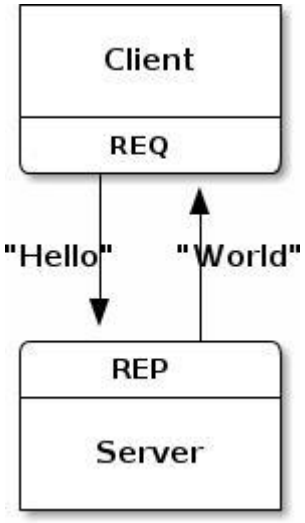
Message receiver subscribes message "Hello" and inverts this message when it receives the message covering the information "Hello".

Example code:

```
########################################
#subtest.py--------subscriber test case
#subscribe message "Hello", when receive message includes "Hello", con
verse this message
########################################
import MessageChannel

def testSubsHandle(data):
    rcvdata = data
    MessageChannel.logging.info("subtest recv data %s" %rcvdata)
    conversestr = rcvdata[::-1]
    MessageChannel.logging.info("converse recv data %s" %conversestr)

if __name__ == "__main__":
    MessageChannel.logconfig("debug")
    testsub = MessageChannel.MessageSubcribe("ipc:///tmp/test-pubsub.i
pc")
    testsub.register_topic("Hello",testSubsHandle)
    MessageChannel.registerSigInt()
    MessageChannel.run()
```

Request/Response Model

Request and response communication model means that a client-side makes a request to a server. E.g., the client-side inquiries current time, and the server replies to the client-side for current time. With the request and response model of asynchronous communication mechanism adopted, the requester can dispose other matters instead of waiting for response of the server after making a request. It can dispose the response after receiving the response from the server. When transmission time expires and the client-side still does not receive confirmation message from the server, the client-side will make a request automatically. Request and response communication diagram is shown below. The client-side sends Hello message to the server, and the server reply to the client-side with “World”.



There is an example that requester sends a message “Hello” to server every 15s and prints received response messages.

Example code:

```
########################################
#reqtest.py--------request test case
#send message "Hello" to the reponse per 15s
########################################
import MessageChannel

def _reqTimeoutHandle(fd, event, obj):
    reqinfo = "Hello"
    req.write(reqinfo)
    MessageChannel.logging.info("request info: %s" %reqinfo)
    MessageChannel.startTimer(reqevt, 15)

def reqMsgHandle(data):
    rcvdata = data
    MessageChannel.logging.info("req handle recvmsg: %s" %rcvdata)

if __name__ == "__main__":
    MessageChannel.logconfig("debug")
    req = MessageChannel.MessageRequest("ipc:///tmp/test-reqrep.ipc",r
eqMsgHandle)
    reqevt = MessageChannel.registerTimer( _reqTimeoutHandle)
    MessageChannel.startTimer(reqevt, 1)
    MessageChannel.registerSigInt()
    MessageChannel.run()
```

Example codes of response, responser sends a message “World” after receiving the message “Hello” from requester.

Example code:

```
########################################
#reptest.py------reponse test case
#recv "Hello", reponse "World" to request
########################################
import MessageChannel
def repMsgHandle(data):
    repdata = data
    MessageChannel.logging.info("rep handle recv: %s" %repdata)
    if repdata == "Hello":
        rep.write("World")

if __name__ == "__main__":
    MessageChannel.logconfig("debug")
    rep = MessageChannel.MessageReponse("ipc:///tmp/test-reqrep.i
pc",repMsgHandle)
    MessageChannel.registerSigInt()
    MessageChannel.run()
```

RouterInfo Device Information Module

RouterInfo module is operation assemble which gets router device information, including cellular module information, GPS information, router firmware version information, etc. See the following table for specific functions. Wherein function return value is JSON.

Function Name	Function Description	Function Return Value
get_firmware_info()	Get router firmware version information	"version_info"
Get_cellular_info()	Get cellular module information	"cellular_info"
Get_gps_info()	Get GPS information	"gps_info"
Get_io_info()	Get IO state information	"io_info"
Get_interface_info()	Get information of all interfaces	"nterface_info"
Get_mac_addr()	Get router MAC address	"mac_info"
Get_conncet_device()	Get information of the device connected with router	"device_info"
Get_routing_info()	Get routing information of router	"routing_info"
Get_dhcp_device()	Get information of the device managed by router DHCP	"dhcp_info"

JSON format is used for function return value, {"return value namen:[key:value,….]}

version_info

Example

{"version_info":{

"lang": "English",

"hostname":"Router",

"timezone": "UTC-8",

"model_name": "915P",

"oem_name": "InHand",

"bootloader":"2011.09.r7903",

"serial_number":"RP9151510285927 ",

"product_number":"PS08-W-S-GPS",

"description":"www.InHand.com.cn",

"encrypt_passwd":"no"

}}

Key	Type	Description
"lang"	Character string	Language currently used by Router Web and CI
"hostname"	Character string	Router hostname
"timezone"	Character string	Router timezone
"model_name"	Character string	Router model
"oem_name"	Character string	Router OEM approved information
"bootloader"	Character string	Router bootloader version number
"serial_number"	Character string	Product serial number of router
"product_number"	Character string	Product model of router

"description"	Character string	Equipment description
"encrypt_passwd"	Character string	Whether encrypted storage is needed by secret key in router configuration files

cellular_info

Example

{"cellular-info":{
"imei": ""
"imsi": "",
"phonenum":
"", "siglevel": "0",
"dbm": 113,
"regstatus":
"0", "network_type": "",
"submode_name": "",
"current_sim": "SIM 1",
"current_operator": "",
"lac": "",
"cellid": "",
"nai_no": "-",
"nai_pwd": "-"
}}

Key	Type	Description
"imei"	Character string	IME code of cellular module
"imsi"	Character string	IMSI code of cellular module
"phonenum"	Character string	Phone number of current dialling SIM card
"siglevel"	Character string	Signal value of cellular module
"dbm"	Character string	Signal strength of cellular module
"regstatus"	Character string	Set forward registration status
"network_ype"	Character string	Wireless network type, e.g. 2G/3G/4G
"submode_name"	Character string	Name of wireless network type, e.g. GPRS/TDD-LTE, etc.
"current_sim"	Character string	Current dialling SIM card
"current_operator"	Character string	Name of network operator
"lac"	Character string	Zone bit code of wireless network site
"nai_no"	Character string	Wireless network access identification
"nai_pwd"	Character string	Wireless network access password

gps_info

Example

{" gps_info":{

"gps_time": "2016-04-28 18:58:28.000",

"latitude": "40&deg00.6371 N",

"longitude": ""116&deg28.1155 E",

"speed": "0.00 Knots (1knot = 1.85km/h)"

}}

Key	Type	Description
"gps_time"	Character string	GPS time, current UTC time
"latitude"	Character string	Latitude, N: north latitude, S: south latitude
"longitude"	Character string	Longitude, E: east longitude, W: west longitude
"speed"	Character string	Current speed

io_info

Example

{"io_info":{

"input":0,

"output": "on"

}}

Key	Type	Description
"input"	Integral type	Display the state of input
"output"	Character string	Display the state of output

InHand Networks

InHand Networks provides reliable, secure and intelligent M2M solution for electric power, industrial automation, commercial and medical devices. InHand Networks is recognized by world class customers and partners and expanding with intensive investments in research and development.

InHand Networks has become leader in industrial grade network technology by providing industrial cellular routers, industrial Ethernet switches, wireless sensor network devices and cloud based M2M platforms.

Connecting devices, enabling services.

InHand Networks

3900 Jermantown Rd., Suite 150
Fairfax, VA 22030
USA
T: +1-703-348-2988
F:+1-703-348-2988
www.inhandnetworks.com
Inquiry: info@inhandnetworks.com
Technical Support: support@inhandnetworks.com