# 라이브러리 설명

udp 를 통하여 간단한 ipc 구현

# 기존 파일소켓(ex. logd)과 다른사항

* 기존의 파일 소켓의 경우 network layer 단계에서 network hw layer까지 내려가지 않으므로 속도가 빠르다.
* 해당 소스는 실제 udp 소켓을 쓰기 때문에 network hw layer까지 내려가므로 아무래도 파일 소켓보다는 약간 성능상의 이슈는 있을 것이다. 하지만, 데이터량도 초당 수십 kb 내외의 크기도 작고, 127.0.0.1 의 localhost 대역을 쓰기 때문에 크게 문제 될 것 같지는 않음.
  + 파일 소켓은 broadcast 가 되지 않는다. 때문에 기존 파일 소켓은 여러 클라이언트에 보내기 위해서는 클라이언트 소켓에 대한 모든 정보를 알고 있어야 했다.
  + udp 의 경우 명시된 port 로 broadcast 가 가능하기 때문에 클라이언트의 정보는 없이 전송 가능하다.

# api 기본 설명

## Port 를 기본으로 통신

* 일반적인 udp server / client 통신과 같으므로 port 번호만 알면된다.
* mds\_udp\_ipc.h 에 기본적으로 포트를 명시해 놓았다.

include/mdsapi/mds\_udp\_ipc.h : 현재 사용중인 기본 포트

#define UDP\_IPC\_PORT\_\_TEST\_APP 30991

#define UDP\_IPC\_PORT\_\_CHK\_GLOBAL\_MEM 30992

#define UDP\_IPC\_PORT\_\_CHK\_APP\_MEM 30993

* + UDP\_IPC\_PORT\_\_TEST\_APP : 셈플 어플리케이션에서 사용하는 포트
  + UDP\_IPC\_PORT\_\_CHK\_GLOBAL\_MEM : global 메모리 변경 시 broadcast 하는 포트
  + UDP\_IPC\_PORT\_\_CHK\_APP\_MEM : 각 app 메모리 변경 시 broadcast 하는 포트

## Port 의 최대 갯수

* 포트는 사용자가 마음대로 추가 할 수 있도록 되어 있다.
  + 이때 최대 10개까지 지원 가능 가능
  + 10개 이상 지원하기 위해서는 다음의 define 을 고친다.

include/mdsapi/mds\_udp\_ipc.h : 최대 지원socket count

#define MAX\_REUSE\_SOCK\_CNT 10

reuse socket

* 최대 지원 port 개수를 정해 놓은 이유는 socket 의 재사용 때문.
* udp client api 구현 시 socket 의 open / close 을 최대한 자제하기 위하여 한번 open 한 socket 을 reuse 하게 구현
* 이때 reuse 를 위한 소켓을 관리하는 최대갯수가 10개이다.
* 매번 open / close 를 할지, 말지는 다음의 feature으로 설정 가능
  + udp\_ipc\_clnt.c >> #define REUSE\_CLNT\_SOCK

# 서버 API

## 관련 API

### 서버시작 API

서버시작 API

void udp\_ipc\_server\_start(int port\_num, int (\*msg\_recv\_proc)(const unsigned char\* recv\_msg, const int recv\_msg\_len, unsigned char\* resp\_msg, int\* resp\_msg\_len));

* 함수 설명
  + 해당 포트 번호로 수신되는 데이터 처리 서버
  + 내부적으로 thread 생성 후 수신 대기
  + 데이터수신 시 msg\_recv\_proc() 함수 포인터 호출
    - 서버에서 receive timeout 일때도 해당 함수 포인터가 호출된다.
  + 클라이언트 응답시 broad cast 가 아니라.. 수신된 클라이언트에게만 응답한다.
  + 응답 시 nowait 로 전송하기 때문에 block 이 걸리지 않는다.
* Return
  + 없음
* argument
  + port\_num : 수신 port number
  + msg\_recv\_proc : 데이터 수신시 호출될 함수포인터

int msg\_recv\_proc\_test(const char\* recv\_msg, const int recv\_msg\_len, unsigned char\* resp\_msg, int\* resp\_msg\_len)

* + - recv\_msg : 클라이언트로부터 수신된 메시지
      * **null 일 경우 time out 으로 인한 호출**
    - recv\_msg\_len : 클라이언트로부터 수신된 메시지 길이
      * **0 일 경우 time out 으로 인한 호출**
    - resp\_msg : 클라이언트 응답용 버퍼
      * 클라이언트에게 응답할 메시지가 있는 경우 버퍼를 넘긴다.
      * **Null 로 넘길 경우 응답하지 않는다.**
    - resp\_msg\_len : 클라이언트 응답용 버퍼 길이
      * 클라이언트에게 응답할 메시지가 있는 경우 버퍼의 길이를 넘긴다.
      * **0 으로 넘길 경우 응답하지 않는다.**

예제소스

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> // sleep()

#include <mdsapi/mds\_udp\_ipc.h>

int msg\_recv\_proc\_test(const unsigned char\* recv\_msg, const int recv\_msg\_len, unsigned char\* resp\_msg, int\* resp\_msg\_len)

{

if (( recv\_msg\_len == 0 ) || (recv\_msg == NULL))

{

printf("recv proc timeout....\r\n");

return 0;

}

printf("msg\_recv\_proc\_test call \r\n");

printf(" - recv\_msg [%s] / [%d]\r\n", recv\_msg, recv\_msg\_len);

// case 1 resp to clnt

strcpy((char\*)resp\_msg, "this msg is resp");

\*resp\_msg\_len = strlen("this msg is resp");

// case 2 not resp to clnt

/\*

\*resp\_msg\_len = 0;

resp\_msg = NULL;

\*/

return 0;

}

int main(void)

{

//unsigned char result\_buff[128] ={0,};

// test case 1 : use thread

udp\_ipc\_server\_start(UDP\_IPC\_PORT\_\_CHK\_APP\_MEM, msg\_recv\_proc\_test);

// test case 2 : no use thread

/\*

if ( receive\_data\_timeout(UDP\_IPC\_PORT\_\_TEST\_APP, 30, result\_buff, 128) > 0 )

printf("result\_buff is [%s]\r\n", result\_buff);

else

printf("receive timeout\r\n");

\*/

while(1)

{

sleep(1);

}

printf("program bye bye..\r\n");

return 0;

}

### 서버종료API

서버종료API

void udp\_ipc\_server\_end(int port\_num);

* 함수 설명
  + 해당 포트 번호로 수신되는 데이터 처리 서버 종료
  + Thread 종료
* Return
  + 없음
* argument
  + port\_num : 수신 port number

### 데이터 수신 API

데이터 수신 AI

int udp\_ipc\_data\_recv\_timeout(int port\_num, int timeout\_sec, unsigned char\* buff, int buff\_len);

* 함수설명
  + 해당 포트로 데이터를 수신을 기다린다.
* Return
  + -1 : timeout 혹은 수신실패
  + 0이상값 : 수신된 데이터 크기
* Argument
  + port\_num : 수신을 기다리는 포트
  + timeout\_sec : timeout 값
  + buff : 데이터 수신 버퍼
  + buff\_len : 데이터 수신 버퍼 크기

# 클라이언트 API

## 데이터 전송 API

데이터 전송API

void udp\_ipc\_broadcast\_send(int port\_num, const unsigned char\* send\_data, const int send\_data\_size);

* 함수설명
  + 해당 포트로 데이터 broadcast 한다.
    - 해당 port 를 open / wait 하고 있는 모든 server 에게 데이터 전송
  + Nowait 옵션으로 block 없이 바로 리턴한다.
  + Broadcast 이기 때문에 정상 전송에 대한 확인 불가.
* Return
  + 없음
* Argument
  + port\_num : broadcast 할 포트
  + send\_data : 데이터 전송할 버퍼
  + send\_data\_size : 데이터 전송할 버퍼 사이즈

예제소스

udp\_ipc\_broadcast\_send(UDP\_IPC\_PORT\_\_CHK\_APP\_MEM, "testmsg", strlen("testmsg"));

## 데이터 전송/수신 API

데이터 전송API

int udp\_ipc\_broadcast\_send\_recv(int port\_num, int time\_out, const unsigned char\* send\_data, const int send\_data\_size, unsigned char\* recv\_buff, int recv\_buff\_size);

* 함수설명
  + 해당 포트로 데이터를 broadcast 하고 데이터를 수신응답을 기다린다.
    - 해당 port 를 open / wait 하고 있는 모든 server 에게 데이터 전송
    - 만약 여러 서버에서 데이터를 동시 전송한다면 첫째 데이터만 응답 받음
  + Broadcast 이기 때문에 정상 전송에 대한 확인 불가.
* Return
  + -1 : timeout 혹은 수신실패
  + 0이상값 : 수신된 데이터 크기
* Argument
  + port\_num : broadcast 할 포트
  + time\_out : 데이터 수신 timeout 값
  + send\_data : 데이터 전송할 버퍼
  + send\_data\_size : 데이터 전송할 버퍼 사이즈
  + recv\_buff : 데이터 수신용 버퍼
  + recv\_buff\_size : 데이터 수신용 버퍼 사이즈

예제소스

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> // sleep()

#include <mdsapi/mds\_udp\_ipc.h>

int main()

{

int i = 10;

int send\_len = 0;

unsigned char send\_buff[IPC\_BUFF\_SIZE] = {0,};

unsigned char recv\_buff[IPC\_BUFF\_SIZE] = {0,};

while (i--)

{

memset(send\_buff, 0x00, IPC\_BUFF\_SIZE);

send\_len = sprintf(send\_buff,"hello msg -> [%d]", i);

udp\_ipc\_broadcast\_send\_recv(UDP\_IPC\_PORT\_\_TEST\_APP, 10, send\_buff, send\_len ,recv\_buff, IPC\_BUFF\_SIZE) ;

printf("[UDP-CLNT-TEST-APP] clnt recv msg [%s]\r\n", recv\_buff);

}

}