

● リスト4

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

int main() {
    int sockfd;
    int client_sockfd;
    struct sockaddr_in addr;
    fd_set rfd;
    int i;

    socklen_t len = sizeof( struct sockaddr_in );
    struct sockaddr_in from_addr;

    char buf[1024];

    // 受信バッファ初期化
    memset( buf, 0, sizeof( buf ) );

    // ソケット生成
    if( ( sockfd = socket( AF_INET, SOCK_STREAM, 0 ) ) < 0 ) {
        perror( "socket" );
    }

    // 待ち受け用IP・ポート番号設定
    addr.sin_family = AF_INET;
    addr.sin_port = htons( 1234 );
    addr.sin_addr.s_addr = INADDR_ANY;

    // バインド
    if( bind( sockfd, (struct sockaddr *)&addr, sizeof( addr ) ) < 0 ) {
        perror( "bind" );
    }

    // 接続要求待ち
    if( listen( sockfd, 10 ) < 0 ) {
        perror( "listen" );
    }

    // クライアントからの接続受け入れ
```

```

if( ( client_sockfd = accept( sockfd, (struct sockaddr *)&from_addr, &len ) ) < 0 ) {
    perror( "accept" );
}

// 受信
int rsize;
while( 1 ) {
    FD_ZERO(&rfdset);
    FD_SET(client_sockfd, &rfdset);
    if (select((client_sockfd + 1), &rfdset, NULL, NULL, NULL) < 1) {
        perror( "select" );
    }

    rsize = recv( client_sockfd, buf, sizeof( buf ), 0 );

    if ( rsize == 0 ) {
        sleep( 10 );
        continue;
    } else if ( rsize == -1 ) {
        perror( "recv" );
    } else {
        printf( "receive(%d):%02x%02x%02x%02x %02x%02x%02x%02x %02x%02x%02x%02x %02x%02x%02x%02x\n", rsize,
            buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7],
            buf[8], buf[9], buf[10], buf[11], buf[12], buf[13], buf[14], buf[15]);
        sleep( 1 );

        // data update
        for (i = 0 ; i < rsize ; i++)
        {
            buf[i] += 0x10;
        }

        // 応答
        printf( "send:%s\n", buf );
        write( client_sockfd, buf, rsize );
    }
}

// ソケットクローズ
close( client_sockfd );
close( sockfd );
return 0;
}

```

- リスト5

```
#include "nx_api.h"

static void NetXduoTest(int nTestNo);
extern NX_IP nx_ip;
extern NX_PACKET_POOL nx_pool;

static void NetXduoTest(int nTestNo)
{
    NX_TCP_SOCKET    TcpSocket;
    NXD_ADDRESS      serverIp;
    UINT status;
    NX_PACKET        *sendData;
    NX_PACKET        *rcvData;
    UCHAR            sendDummyData[16];
    UCHAR            rcvDummyData[16];
    ULONG            rcvSize;
    int              i;

    /* Create TCP/IP Socket */
    status = nx_tcp_socket_create(&nx_ip, &TcpSocket, "ConnectLinuxSocket",
        NX_IP_NORMAL, NX_DONT_FRAGMENT, NX_IP_TIME_TO_LIVE,
        1024, NX_NULL, NX_NULL);
    if (status != NX_SUCCESS)
    {
        printf("NetXduo nx_tcp_socket_create() failed.(ret=%x)\r\n", status);
        return;
    }

    /* Bind TCP/IP Port*/
    status = nx_tcp_client_socket_bind(&TcpSocket, NX_ANY_PORT, 300);
    if (status != NX_SUCCESS)
    {
        nx_tcp_socket_delete(&TcpSocket);
        printf("NetXduo nx_tcp_client_socket_bind() failed.(ret=%x)\r\n", status);
        return;
    }

    /* Connect Linux Server */
    serverIp.nxd_ip_version = NX_IP_VERSION_V4;
    serverIp.nxd_ip_address.v4 = IP_ADDRESS(192,168,0,20);
    status = nxd_tcp_client_socket_connect(&TcpSocket, &serverIp, 1234, 300);
```

```

if (status != NX_SUCCESS)
{
    nx_tcp_client_socket_unbind(&TcpSocket);
    nx_tcp_socket_delete(&TcpSocket);
    printf("NetXduo nx_tcp_client_socket_connect() failed.(ret=%x)\r\n", status);
    return;
}

/* Get Send data packet area */
status = nx_packet_allocate(&nx_pool, &sendData, NX_TCP_PACKET, NX_WAIT_FOREVER);
if (status != NX_SUCCESS)
{
    nx_tcp_socket_disconnect(&TcpSocket, 300);
    nx_tcp_client_socket_unbind(&TcpSocket);
    nx_tcp_socket_delete(&TcpSocket);
    printf("NetXduo nx_packet_allocate() failed.(ret=%x)\r\n", status);
    return;
}

/* Set send data */
for (i = 0 ; i < sizeof(sendDummyData); i++)
{
    sendDummyData[i] = (i + 1);
}
nx_packet_data_append(sendData, sendDummyData, sizeof(sendDummyData),
    &nx_pool, NX_WAIT_FOREVER);

/* Send dummy data */
status = nx_tcp_socket_send(&TcpSocket, sendData, NX_WAIT_FOREVER);
if (status != NX_SUCCESS)
{
    nx_tcp_socket_disconnect(&TcpSocket, 300);
    nx_tcp_client_socket_unbind(&TcpSocket);
    nx_tcp_socket_delete(&TcpSocket);
    printf("NetXduo nx_tcp_socket_send() failed.(ret=%x)\r\n", status);
    return;
}

/* Recv dummy data */
status = nx_tcp_socket_receive(&TcpSocket, &rcvData, NX_WAIT_FOREVER);
if (status != NX_SUCCESS)
{
    nx_tcp_socket_disconnect(&TcpSocket, 300);
    nx_tcp_client_socket_unbind(&TcpSocket);

```

```

    nx_tcp_socket_delete(&TcpSocket);
    printf("NetXduo nx_tcp_socket_receive() failed.(ret=%x)\r\n", status);
    return;
}

/* Get rcv data */
status = nx_packet_data_retrieve(rcvData, rcvDummyData, &rcvSize);
if (status != NX_SUCCESS)
{
    nx_tcp_socket_disconnect(&TcpSocket, 300);
    nx_tcp_client_socket_unbind(&TcpSocket);
    nx_tcp_socket_delete(&TcpSocket);
    printf("NetXduo nx_packet_data_retrieve() failed.(ret=%x)\r\n", status);
    return;
}

/* Disconnect to Linux Server */
nx_tcp_socket_disconnect(&TcpSocket, 300);

/* Unbind Network port */
nx_tcp_client_socket_unbind(&TcpSocket);

/* Delete TCP/IP Socket */
nx_tcp_socket_delete(&TcpSocket);
}

```

● リスト6

```

CHAR    *pointer;
UINT    status;
NX_HTTP_SERVER    httpServer;

pointer = (CHAR *) first_unused_memory;

/* Create the HTTP Server */
status = nx_http_server_create(&httpServer, "My HTTP Server", &nx_ip,
    &ram_disk, pointer, 2048, &nx_pool, authentication_check, http_request_notify);
if (status)
{
    printf("NetXduo nx_http_server_create() failed.(ret=%x)\r\n", status);
    return;
}

```

```

status = nx_http_server_start(&httpServer);
if (status != NX_SUCCESS)
{
    printf("NetXduo nx_http_server_start() failed.(ret=%x)\r\n", status);
    return;
}

```

● リスト7

```

static UINT authentication_check(NX_HTTP_SERVER *server_ptr, UINT request_type,
    CHAR *resource, CHAR **name, CHAR **password, CHAR **realm)
{
    *name = "name";
    *password = "password";
    *realm = "NetX Duo HTTP demo";
    return(NX_HTTP_BASIC_AUTHENTICATE);
}

```

● リスト8

```

static UINT http_request_notify(NX_HTTP_SERVER *server_ptr, UINT request_type,
    CHAR *resource, NX_PACKET *recv_packet_ptr)
{
    if ((request_type == NX_HTTP_SERVER_GET_REQUEST) &&
        (strcmp(resource, "/test.htm") == 0))
    {
        nx_http_server_callback_data_send(server_ptr,
            "HTTP/1.0 200 \r\nContent-Length:103\r\nContent-Type: text/html\r\n\r\n",
            63);
        nx_http_server_callback_data_send(server_ptr, "<HTML>\r\n<HEAD><TITLE>NetX HTTP Test </TITLE></HEAD>\r\n<BODY>\r\n<H1>NetX Test Page</H1>\r\n</BODY>\r\n</HTML>\r\n", 103);
        return(NX_HTTP_CALLBACK_COMPLETED);
    }
    return(NX_SUCCESS);
}

```