A76xx Series Open SDK_ TCP UDP 网络开发_应用指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_TCP/UDP网络开发_应用指导
版本：	V1.00
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

Version History

Version	Date	Owner	What is new
V1.00	2022-11-18		第一版

About this Document

本文档适用于 A1803S open 系列、A1603 open 系列、A1606 open 系列。

目录

缩略语

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hyper Text Transport Protocol
MIME	Multipurpose Internet Mail Extensions
URL	Uniform Resource Locator
WWW	World Wide Web
IP	Internet Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
MQTT	Message Queuing Telemetry Transport

1TCP/UDP 套接字开发

1.1 套接字介绍

所谓套接字(Socket)，就是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象，可以看成在两个程序进行通讯连接中的一个端点，一个程序将一段信息写入 Socket 中，该 Socket 将这段信息发送给另外一个 Socket 中。

1.2 套接字类型

套接字类型指的是套接字的数据传输方式，通过 socket 函数的第一个参数:传递，只有这样才能决定创建的套接字的数据传输方式。

流式套接字（SOCK_STREAM ）

如果使用的是流式套接字，意味着它使用了 TCP 协议，TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议，能够保证数据传输的正确性。

数据报套接字 （SOCK_DGRAM ）

如果使用的是数据报套接字，意味着它使用了 UDP 协议。它是一种不可靠的，不按顺序的，以数据的高速传输为目的的套接字。

原始套接字（SOCK_RAM）

原始套接字允许对低层协议如 IP 或 ICMP 直接访问，主要用于新的网络协议的测试等。

2API 介绍

2.1 sAPI_TcpipGetErrno

获取 tcpip 错误码

原型: :	INT32 sAPI_TcpipGetErrno(void);
参数: :	无
返回值: :	lwip_errno
头文件: :	#include "simcom_tcpip.h"

2.2 sAPI_TcpipPdpActive

激活 tcpip 上下文

原型: :	INT32 sAPI_TcpipPdpActive(int cid, int channel);
参数: :	[in] cid: 指定特定 PDP 上下文定义的数值参数:。 TCPIP cid 是 1. [in] channel: 数据传输通道 0 - serial port 1 - usb
返回值: :	lwip_errno
头文件:	#include "simcom_tcpip.h"

2.3 sAPI_TcpipPdpDeactive

撤销 tcpip 上下文

原型:	INT32 sAPI_TcpipPdpDeactive(int cid, int channel);
参数:	[in] cid: 指定特定 PDP 上下文定义的数值参数:。 TCPIP cid 是 1. [in] channel: 数据传输通道 0 - serial port 1 - usb
返回值:	0：撤销成功 -1：撤销失败
头文件:	#include "simcom_tcpip.h"

2.4 sAPI_TcpipPdpDeactiveNotify

当 PDP 被撤销时通知。

原型:	INT32 sAPI_TcpipPdpDeactiveNotify(sMsgQRef msgQ);
参数:	[out] msgQ: 结果将由 msgQ 返回
返回值:	0：通知成功 -1：通知失败
头文件:	#include "simcom_tcpip.h"

2.5 sAPI_TcpipGetSocketPdpAddr

获取 tcpip PDP 地址

原型:	INT32 sAPI_TcpipGetSocketPdpAddr(int cid, int channel, struct simcom_ip_info *info);
参数:	[in] cid: 指定特定 PDP 上下文定义的数值参数:。 TCPIP cid 是 1. [in] channel: 数据传输通道 [in]info：指向包含 IP 地址信息的结构的指针。
返回值:	0：获取成功 -1：获取失败
头文件:	#include "simcom_tcpip.h"

2.6 sAPI_TcpipSocket

创建用于通信的端点并返回描述符

原型:	INT32 sAPI_TcpipSocket(INT32 domain, INT32 type, INT32 protocol);
参数:	[in] domain:协议族 [in] type:套接字类型 [in] protocol: 指定要与套接字一起使用的特定协议
返回值:	如果成功，则返回新套接字的文件描述符。 -1：创建 sockt 套接字失败
头文件:	#include "simcom_tcpip.h"

2.7 sAPI_TcpipBind

将名称绑定到套接字

原型:	INT32 sAPI_TcpipBind(INT32 sockfd, const SCsockAddr *addr, UINT32 addrlen);

参数:	[in] sockfd: 文件描述符 [in] addr: 分配由 addr 指定的地址 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	0：将名称绑定到套接字成功。 -1：绑定名称到套接字失败。
头文件:	#include "simcom_tcpip.h"

2.8 sAPI_TcpipListen

监听套接字上的连接

原型:	INT32 sAPI_TcpipListen(INT32 sockfd, INT32 backlog);
参数:	[in] scokfd: 文件描述符 [in] backlog: 定义挂起连接队列的最大长度
返回值:	0：成功侦听套接字上的连接 -1：监听 socket 连接失败。
头文件:	#include "simcom_tcpip.h"

2.9 sAPI_TcpipAccept

接受套接字上的连接

原型:	INT32 sAPI_TcpipAccept(INT32 sockfd, SCsockAddr *addr, UINT32 *addrlen);
参数:	[in] scokfd:文件描述符 [in] addr: 地址 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	如果成功，返回一个非负整数，它是被接受的套接字的描述符 -1:失败。
头文件:	#include "simcom_tcpip.h"

2.10 sAPI_TcpipConnect

在套接字上发起连接

原型:	INT32 sAPI_TcpipConnect(INT32 sockfd, const SCsockAddr *addr, UINT32 addrlen);
参数:	[in] scokfd:文件描述符 [in] addr: 地址 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	0：在套接字上初始化连接成功 -1：在套接字上发起连接失败
头文件:	#include "simcom_tcpip.h"

2.11 sAPI_TcpipSend

在套接字上发送消息。api 只能在套接字处于连接状态时使用

原型:	INT32 sAPI_TcpipSend(INT32 sockfd, const void *buf, INT32 len, INT32 flags);
参数:	[in] scokfd: 文件描述符 [in] buf: 指定发送 buf [in] len: 指定 buf 的大小，以字节为单位 [in] flags: 一组标志，指定进行呼叫的方式。通常设置为 0
返回值:	如果成功，返回发送的字符数。 0：对端正常关机 -1：在套接字上发送消息失败
头文件:	#include "simcom_tcpip.h"

2.12 sAPI_TcpipRecv

从套接字接收消息

原型:	INT32 sAPI_TcpipRecv(INT32 sockfd, void *buf, INT32 len, INT32 flags);
参数:	[in] scokfd: 文件描述符 [in] buf: 指定接收 buf [in] len: 指定接收 buf 的大小，以字节为单位 [in] flags: 一组标志，指定进行呼叫的方式。通常设置为 0
返回值:	成功时，返回接收到的字节数 0：对端正常关机 -1：从套接字接收消息失败
头文件:	#include "simcom_tcpip.h"

2.13 sAPI_TcpipSendto

在套接字上发送消息

原型:	INT32 sAPI_TcpipSendto(INT32 sockfd, const void *buf, INT32 len, INT32 flags, const SCsockAddr *dest_addr, UINT32 addrlen);
参数:	[in] scokfd: 文件描述符 [in] buf: 指定发送 buf [in] len: 指定 buf 的大小，以字节为单位 [in] flags: 一组标志，指定进行呼叫的方式。通常设置为 0 [in] dest_addr: 地址 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	如果成功，返回发送的字符数。 -1: socket 发送消息失败
头文件:	#include "simcom_tcpip.h"

2.14 sAPI_TcpipRecvfrom

从套接字接收消息

原型:	INT32 sAPI_TcpipRecvfrom(INT32 sockfd, void *buf, INT32 len, INT32 flags, SCsockAddr *src_addr, UINT32 *addrlen);
参数:	[in] scokfd: 文件描述符 [in] buf: 指定接收 buf [in] len: 指定 buf 的大小，以字节为单位 [in] flags: 指定要与套接字一起使用的特定协议。通常设置为 0 [in] src_addr: 地址 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	成功时，返回接收到的字节数。 -1:从 socket 接收消息失败
头文件:	#include "simcom_tcpip.h"

2.15 sAPI_TcpipClose

关闭一个文件描述字

原型:	INT32 sAPI_TcpipClose(INT32 fd);
参数:	[in] sockfd: 文件描述符
返回值:	0:关闭文件描述符成功 -1:关闭文件描述符失败
头文件:	#include "simcom_tcpip.h"

2.16 sAPI_TcpipShutdown

关闭全双工连接的一部分

原型:	INT32 sAPI_TcpipShutdown(INT32 sockfd, INT32 how);
参数:	[in] sockfd: 文件描述符 [in] how: 指定通信语义。
返回值:	0: 成功关闭了全双工连接的一部分 -1: 关闭部分全双工连接失败
头文件:	#include "simcom_tcpip.h"

2.17 sAPI_TcpipGetsockname

函数用于获取与某个套接字关联的本地协议地址

原型:	INT32 sAPI_TcpipGetsockname(INT32 sockfd, SCsockAddr *addr, UINT32 *addrlen);
参数:	[in] sockfd: 文件描述符

	[out] addr: 获取到的地址信息 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	0: 获取套接字名称成功 -1: 获取套接字名称失败
头文件:	#include "simcom_tcpip.h"

2.18 sAPI_TcpipGetpeername

函数用于获取与某个套接字关联的外地协议地址

原型:	INT32 sAPI_TcpipGetpeername(INT32 sockfd, SCsockAddr *addr, UINT32 *addrlen);
参数:	[in] sockfd: 文件描述符 [out] addr: 获取到的地址信息 [in] addrlen: 指定 addr 的大小，以字节为单位
返回值:	0: 获取已连接的对等端套接字名称成功 -1: 获取连接的对端套接字名称失败
头文件:	#include "simcom_tcpip.h"

2.19 sAPI_TcpipGetsockopt

获取任何类型、任何状态套接字接口的选项的当前值，并将结果存储在 OptVAL 中

原型:	INT32 sAPI_TcpipGetsockopt(INT32 sockfd, INT32 level, INT32 optname, void *optval, UINT32 *optlen);
参数:	[in] scokfd: 文件描述符 [in] level: 协议层次。 [in] optname: 选项名称 [in] optval: 获取到的选项的值 [in] optlen: 选项值的长度
返回值:	0: 成功获取套接字上的选项 -1: 在套接字上获取选项失败
头文件:	#include "simcom_tcpip.h"

2.20 sAPI_TcpipSetsockopt

在套接字上设置选项

原型:	INT32 sAPI_TcpipSetsockopt(INT32 sockfd, INT32 level, INT32 optname, const void *optval, UINT32 optlen);
参数:	[in] scokfd: 文件描述符 [in] level: 协议层次。 [in] optname: 选项名称 [in] optval: 获取到的选项的值 [in] optlen: 选项值的长度
返回值:	0: 成功设置套接字上的选项

	-1: 在套接字上设置选项失败
头文件:	#include "simcom_tcpip.h"

2.21 sAPI_TcpipIoctlsocket

控制 socket 接口的模式。可以在任何状态下使用的任意一组接口。它用于获取与套接字接口相关的操作参数:，而不考虑具体的协议或通信子系统。

原型:	INT32 sAPI_TcpipIoctlsocket(INT32 fd, INT32 level, void *argp);
参数:	[in] fd: 文件描述符。 [in] level: 对套接字 fd 的操作命令。 [in] argp: 指向操作命令参数:的指针
返回值:	0: 控制装置成功 -1: 无法控制设备
头文件:	#include "simcom_tcpip.h"

2.22 sAPI_TcpipGethostbyname

为给定的主机名返回 hostent 类型的结构

原型:	SChostent * sAPI_TcpipGethostbyname(const INT8 *name);
参数:	[in] **name**: 主机名
返回值:	如果成功，返回 hostent 类型的结构。 如果出错，则返回 NULL 指针
头文件:	#include "simcom_tcpip.h"

2.23 sAPI_TcpipSelect

该函数允许程序监视多个文件描述符，等待一个或多个文件描述符“准备好”进行某种 I/O 操作。

原型:	INT32 sAPI_TcpipSelect(INT32 nfds, SCfdSet *readfds, SCfdSet *writefds, SCfdSet *exceptfds, SCtimeval *timeout);
参数:	[in] **nfds**:最大文件描述符加 1 [in] **readfds**: （可选）指针，指向一组等待可读性检查的套接口。 [in] **writefds**: （可选）指针，指向一组等待可写性检查的套接口。 [in] **exceptfds**: （可选）指针，指向一组等待错误检查的套接口。. [in] **timeout**: 本函数最多等待时间，对阻塞操作则为 NULL。
返回值:	成功后，返回三个返回的描述符集中包含的文件描述符的数量(即 readfds, writefds, exceptfds 中设置的总位数);如果超时在任何事情发生之前就过期了，则该值可能为零。 如果出错，则返回 NULL 指针 -1:无法监控多个文件描述符
头文件:	#include "simcom_tcpip.h"

2.24 sAPI_TcpipInet_addr

将 Internet 主机地址 cp 从 IPv4 数字点表示法转换为网络字节顺序的二进制数据

原型:	UINT32 sAPI_TcpipInet_addr(const INT8 *cp);
参数:	[in] **cp**: 因特网主机地址。
返回值:	0:转换 Internet 主机地址成功。 -1:转换 Internet 主机地址失败
头文件:	#include "simcom_tcpip.h"

2.25 sAPI_TcpipHtons

将无符号短整数 hostshort 从主机字节顺序转换为网络字节顺序

原型:	UINT16 sAPI_TcpipHtons(UINT16 hostshort);
参数:	[in] **hostshort**: 16 位无符号短整数
返回值:	0:表示无符号短整数转换成功 -1:无符号短整数主机短转换失败
头文件:	#include "simcom_tcpip.h"

2.26 sAPI_TcpipNtohs

将无符号短整数 netshort 从网络字节顺序转换为主机字节顺序

原型:	UINT16 sAPI_TcpipNtohs(UINT16 hostshort);
参数:	[in] **hostshort**: 16 位无符号短整数
返回值:	0:无符号短整数 netshort 转换成功 -1:无符号短整数 netshort 转换失败
头文件:	#include "simcom_tcpip.h"

2.27 sAPI_TcpipInet_ntoa

将以网络字节顺序给出的 Internet 主机地址转换为 IPv4 点分十进制记数法的字符串。该字符串在静态分配的缓冲区中返回，后续调用将覆盖该缓冲区。

原型:	INT8 *sAPI_TcpipInet_ntoa(UINT32 in);
参数:	[in] in: 因特网主机地址
返回值:	NULL: Internet 主机地址转换失败 Other: Internet 主机地址转换成功
头文件:	#include "simcom_tcpip.h"

2.28 sAPI_TcpipInet_ntop

将网络二进制结构转换为点分十进制类型地址。。

原型:	INT8 *sAPI_TcpipInet_ntop(INT32 af, const void *src, INT8 *dst, UINT32 size);
参数:	[in] af: 簇地址 [in] src:源地址 [in] dst: 转换数据 [in] size: 转换数据的长度
返回值:	NULL: Internet 主机地址转换失败 Other: Internet 主机地址转换成功
头文件:	#include "simcom_tcpip.h"

2.29 sAPI_TcpipGetSocketErrno

获取 sockfd 的错误码。

原型:	INT32 sAPI_TcpipGetsocketErrno(int sockfd);
参数:	[in] **sockfd**: socket id
返回值:	errno
头文件:	#include "simcom_tcpip.h"

2.30 sAPI_TcpipHtonl

将主机字节顺序转换为网络字节顺序

原型:	UINT32 sAPI_TcpipHtonl(INT32 hostlong);
参数:	[in] **hostlog**: 32 位主机字节数据
返回值:	32 位网络字节数据
头文件:	#include "simcom_tcpip.h"

2.31 sAPI_TcpipGetaddrinfo

主机名到地址解析。

原型:	int *sAPI_TcpipGetaddrinfo(const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res);
参数:	[in] **nodename**: 一个主机名或者地址字符串 [in] **servname**: 端口号 [in] **hints**: 可以是一个空指针，也可以是一个指向某个 addrinfo 结构体的指针

	[in] **res**: 本函数通过 result 指针参数:返回一个指向 addrinfo 结构体链表的指针
返回值:	0: 成功 -1:失败
头文件:	#include "simcom_tcpip.h"

2.32 sAPI_TcpipGetaddrinfo_with_pcid

通过指定 pcid 获得网络地址

原型:	int *sAPI_TcpipGetaddrinfo_with_pcid(const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res, u8_t pcid);
参数:	[in] **nodename**: 一个主机名或者地址字符串 [in] **servname**: 端口号 [in] **hints**: 可以是一个空指针，也可以是一个指向某个 addrinfo 结构体的指针 [in] res: 本函数通过 result 指针参数:返回一个指向 addrinfo 结构体链表的指针 [in]:pcid:指定的 pcid
返回值:	0: 成功 -1:失败
头文件:	#include "simcom_tcpip.h"

2.33 sAPI_TcpipFreeaddrinfo

释放 addrinfo 指针。

原型:	void sAPI_TcpipFreeaddrinfo(struct addrinfo *ati);
参数:	[in] **ati**: addrinfo 指针
返回值:	无
头文件:	#include "simcom_tcpip.h"

2.34 sAPI_TcpipSocket_with_callback

创建用于通信的端点并返回描述符。当套接字的状态发生变化时，会触发一个回调函数

原型:	int sAPI_TcpipSocket_with_callback(int domain, int type, int protocol,socket_callback callback);
参数:	[in] **domain**:地址协议族 [in] **type**:数据传输类型 [in] **protocol**: 指定要与套接字一起使用的特定协议 [in] **callback**:回调函数
返回值:	如果成功，则返回新套接字的文件描述符 -1: 从套接字接收消息失败
头文件:	#include "simcom_tcpip.h"

2.35 sAPI_TcpipInet_pton

将 IPv4 和 IPv6 地址从点分十进制转换为二进制。

原型:	int sAPI_TcpipInet_pton(int af, const char *src, void *dst);
参数:	[in] af: 簇地址 [in] src: 源地址 [in] **dst**: 转换数据
返回值:	NULL: Internet 主机地址转换失败 Other: 成功转换 Internet 主机地址
头文件:	#include "simcom_tcpip.h"

3 变量定义

3.1 套接字宏定义

```
/* Socket protocol types (TCP/UDP/RAW) */
#define SC_SOCK_STREAM     1         //TCP
#define SC_SOCK_DGRAM      2         //UDP
#define SC_SOCK_RAW        3         //RAW


#define SC_AF_INET         2             //ipv4
#define SC_AF_INET6        10            //ipv6
#define SC_INADDR_ANY      ((INT32)0x00000000)

#define SC_SO_BIO          0x100c  /* 设置套接字为阻塞模式*/
#define SC_SO_NBIO         0x100d  /* 设置套接字为非阻塞*/
详细的宏定义请参考 simcom_tcpip.h
```

3.2 SCsockAddrIn

```
typedef struct
{
    UINT8 sin_len;                                  //表示该结构体的长度
    UINT8 sin_family;                               //协议族
    UINT16 sin_port;                               //端口号
    SCinAddr sin_addr;                             //ip 地址结构体
#define SC_SIN_ZERO_LEN 8
    INT8 sin_zero[SC_SIN_ZERO_LEN];          //填充字节数组
} SCsockAddrIn;



typedef struct
{
    UINT32 s_addr;          //表示 32 位的 IP 地址，32 位无符号整型
} SCinAddr;
```

3.3 SCtimeval

```
typedef struct
{
    INT32 tv_sec;       /* 时间单位为秒*/
    INT32 tv_usec;      /* 时间单位为微秒*/
} SCtimeval;
```

3.4 SCaddrinfo

```
typedef struct
{
    UINT8 sa_len;
    UINT8 sa_family;
#if LWIP_IPV6
    UINT8 sa_data[22];
#else /* LWIP_IPV6 */
    UINT8 sa_data[14];
#endif /* LWIP_IPV6 */
} SCsockAddr;

struct SCaddrinfo {
    int             ai_flags;     /*输入模式标志. */
    int             ai_family;    /*套接字地址协议族. */
    int             ai_socktype;  /* 套接字类型*/
    int             ai_protocol;  /* 套接字协议. */
    UINT32          ai_addrlen;   /*套接字地址长度. */
    SCsockAddr *ai_addr;      /*套接字地址. */
    char            *ai_canonname; /*服务位置的规范名称. */
    struct SCaddrinfo  *ai_next;     //指针
};
```

3.5 SChostent

```
typedef struct
{
    INT8  *h_name;     /* 主机名. */
    INT8 **h_aliases;  /*一个指针，指向指向可选主机名的指针数组， 以空指针结束。 */
    INT32    h_addrtype;  /* 地址类型*/
    INT32    h_length;    /* 地址的长度，以字节为单位 */
    INT8 **h_addr_list; /*一个指针，指向指向主机网络地址(按 网络字节顺序)的指针数组，以空指针结束. */
#define h_addr h_addr_list[0] /* 保留过去的用法只是为了相容性
 */
} SChostent;
```

4 Example

```
#include "simcom_tcpip.h"
#include "simcom_debug.h"

INT32 sAPI_TCPIPConnectTcpServerTestMain(INT8*ipstr, UINT16 port, UINT16 localPort);
{
    INT32 sockfd = -1;
    INT32 ret = -1;
    SChostent *host_entry = NULL;
    SCsockAddrIn sa;
    SCsockAddrIn server;
    INT32 keepalive = 1, keepidle = 10, keepcount = 2, keepINT32erval = 10;
    INT8 recvbuf[100] = {0};
    INT32 len = 0;
       INT8 ip[100] ;
    struct simcom_ip_info ip_info = {INVALID, 0, {0}};

    if(-1 == sAPI_TcpipPdpActive(1, 1););
    {
      sAPI_Debug("PDP active err");
      return ret;
    }

    if(-1 == sAPI_TcpipGetSocketPdpAddr(1, 1, &ip_info););
    {
      sAPI_Debug("PDP active err");
      sAPI_TcpipPdpDeactive(1, 1);
      return ret;
    }

    sAPI_Debug("PDP type = %d, ipv4 = %d, ipv6 = %s", ip_info.type, ip_info.ip4, ip_info.ip6);
    sockfd = sAPI_TcpipSocket(SC_AF_INET, SC_SOCK_STREAM, 0);
    if(sockfd < 0);
    {
        sAPI_Debug("create socket err");
        goto err;
    }
```

```
    host_entry = sAPI_TcpipGethostbyname(ipstr);
    if(host_entry == NULL);
    {
        sAPI_Debug("DNS gethostbyname fail");
        goto err;
    }

    sAPI_Debug("DNS gethostbyname, Get %s ip %d.%d.%d.%d\n",
            ipstr, host_entry->h_addr_list[0] [0] & 0xff,
            host_entry->h_addr_list[0] [1] & 0xff, host_entry->h_addr_list[0] [2] & 0xff,
            host_entry->h_addr_list[0] [3] & 0xff);

    ret = sAPI_TcpipSetsockopt(sockfd, SC_SOL_SOCKET, SC_SO_KEEPALIVE, (void *);&keepalive,
sizeof(keepalive););
    if(ret < 0);
        goto err;

    ret = sAPI_TcpipSetsockopt(sockfd, SC_IPPROTO_TCP, SC_TCP_KEEPIDLE, (void*);&keepidle,
sizeof(keepidle););
    if(ret < 0);
        goto err;

    ret = sAPI_TcpipSetsockopt(sockfd, SC_IPPROTO_TCP, SC_TCP_KEEPCNT, (void *);&keepcount,
sizeof(keepcount););
    if(ret < 0);
        goto err;

    ret = sAPI_TcpipSetsockopt(sockfd, SC_IPPROTO_TCP, SC_TCP_KEEPINTVL, (void
*);&keepINT32erval, sizeof(keepINT32erval););
    if(ret < 0);
        goto err;

    ret = sAPI_TcpipGetsockopt(sockfd, SC_IPPROTO_TCP, SC_TCP_KEEPINTVL, (void
*);&keepINT32erval, sizeof(keepINT32erval););
    if(ret < 0);
        goto err;

    if(localPort> 0);
    {
        sa.sin_family       = SC_AF_INET;
```

```
    sa.sin_port         = htons(localPort);
    sa.sin_addr.s_addr  = 0;
    if(sAPI_TcpipBind(sockfd, (const SCsockAddr *);&sa, sizeof(sa);); < 0);
    {
        sAPI_Debug("Bind local port fail errno[%d] ", sAPI_TcpipGetErrno(););
        goto err;
    }
}

server.sin_family = SC_AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr= *(UINT32 *);host_entry->h_addr_list[0] ;


ret = sAPI_TcpipConnect(sockfd, &server, sizeof(SCsockAddr););
if(ret != 0);
{
    sAPI_Debug("connect server fail errno[%d] ", sAPI_TcpipGetErrno(););
    goto err;
}

sAPI_Debug("ret[%d] connect server sucess", ret);

//send hello to server

ret = sAPI_TcpipSend(sockfd, "hello", 5, 0);
if(ret < 0);
{
    sAPI_Debug("send hello to server  fail errno[%d] ", sAPI_TcpipGetErrno(););
    goto err;
}

memset(recvbuf, 0x0, sizeof(recvbuf););

ret = sAPI_TcpipRecv(sockfd, recvbuf, sizeof(recvbuf), 0);
if(ret < 0);
{
    sAPI_Debug("recv from server fail errno[%d] ", sAPI_TcpipGetErrno(););
    goto err;
```

```
    }

    sAPI_Debug("recv SUCESS byte:[%d]  recvbuf[%s] ", ret, recvbuf);

    len = sizeof(sa);
    ret = sAPI_TcpipGetsockname(sockfd, (SCsockAddr *);&sa, &len);
    if(ret < 0);
        goto err;

    memset(ip, 0x0, sizeof(ip););
    if(sAPI_TcpipInetNtop(SC_AF_INET, &sa.sin_addr, ip, sizeof(ip);); == NULL);
        goto err;

    sAPI_Debug("host ip[%s] port[%d] ", ip, sAPI_TcpipNtohs(sa.sin_port););
err:
    if(sockfd>= 0);
    {
        sAPI_TcpipClose(sockfd);
        sAPI_TcpipPdpDeactive(1, 1);
    }

    return ret;

}
void sAPI_TCPIPListenTestMain();
{
    INT32 ret = -1;
    INT32 serverfd = 0, newfd = 0;
    SChostent *host_entry;
    SCsockAddrIn sa;
    SCsockAddrIn cliaddr;
    INT32 reuseport = 1;
    INT32 fdmax = -1;
    SCfdSet read_fds;
    INT32 i = 0;
    INT32 len = 0;

    INT32 serverport = 8888;
    INT32 backlog = 5;
```

```
    struct simcom_ip_info ip_info = {INVALID, 0, {0}};

    if(-1 == sAPI_TcpipPdpActive(1, 1););
    {
      sAPI_Debug("PDP active err");
      return ret;
    }

    if(-1 == sAPI_TcpipGetSocketPdpAddr(1, 1, &ip_info););
    {
      sAPI_Debug("PDP active err");
      sAPI_TcpipPdpDeactive(1, 1);
      return ret;
    }
    serverfd = sAPI_TcpipSocket(SC_AF_INET, SC_SOCK_STREAM, 0);
    if(serverfd < 0);
    {
        sAPI_Debug("create socket err");
        goto err;
    }

    memset(&(sa), 0, sizeof(sa));
    sa.sin_addr.s_addr = sAPI_TcpipInetAddr("127.0.0.1");

    sAPI_Debug("s_addr[%d] ", sa.sin_addr.s_addr);

    if(sa.sin_addr.s_addr == SC_IPADDR_None);
    {
        goto err;
    }
    sa.sin_family      = SC_AF_INET;
    sa.sin_port        =  sAPI_TcpipHtons(serverport);

    ret = sAPI_TcpipSetsockopt(serverfd, SC_SOL_SOCKET, SC_SO_REUSEADDR, &reuseport,
sizeof(reuseport));
    if(ret < 0);
        goto err;



    if(sAPI_TcpipBind(serverfd, (const SCsockAddr *);&sa, sizeof(sa)); < 0);
```

```
{
    sAPI_Debug("Bind local port fail errno[%d] ", sAPI_TcpipGetErrno(););
    goto err;
}

if(sAPI_TcpipListen(serverfd, backlog); < 0);
{
    sAPI_Debug("listen fail errno[%d] ", sAPI_TcpipGetErrno(););
    goto err;
}

SC_FD_ZERO(&read_fds);
SC_FD_SET(serverfd, &read_fds);
fdmax = serverfd+1;
ret = sAPI_TcpipSelect(fdmax, &read_fds, NULL, NULL, NULL);
if(ret == -1 || ret == 0);
{
    sAPI_Debug("select fail or timeout[%d] ", sAPI_TcpipGetErrno(););
    goto err;
}

sAPI_Debug("--- ret[%d] ", ret);

for(i = 0;i < fdmax;i++);
{
    if(!SC_FD_ISSET(i, &read_fds););
    {
        continue;
    }

    if(i == serverfd);
    {
        char ipstr[100] ;
        UINT16 clientport = 0;


        len = sizeof(cliaddr);
        newfd = sAPI_TcpipAccept(serverfd, (SCsockAddr *);&cliaddr, (UINT32 *);&len);
        sAPI_Debug("--- newfd[%d] ", newfd);
```

```
        if(newfd < 0);
        {
            sAPI_Debug("accept fail [%d] ", sAPI_TcpipGetErrno(););
            goto err;
        }

        clientport = sAPI_TcpipNtohs(cliaddr.sin_port);
        memset(ipstr, 0x0, sizeof(ipstr););
        snprintf(ipstr, sizeof(ipstr), "%s", sAPI_TcpipInetNtoa(cliaddr.sin_addr.s_addr););

        sAPI_Debug("new client connect port[%d] ip[%d] newfd[%d] ", clientport, ipstr, newfd);

        ret = sAPI_TcpipSend(newfd, "hello client", strlen("hello client"), 0);
        if(ret < 0);
        {
            sAPI_Debug("send to client fail");
            goto err;
        }

        sAPI_TcpipClose(newfd);
        sAPI_TcpipPdpDeactive(1, 1);
        ret = 0;
      }

   }
 err:

   if(serverfd> 0);
   {
      sAPI_TcpipClose(serverfd);
      sAPI_TcpipPdpDeactive(1, 1);
   }

   return;
}

INT32 sAPI_UDPTestMain(INT32 localport, char *addr, UINT16 port);
{
```

```
INT32 ret = -1;
INT32 sockfd = 0;
SCsockAddrIn sa;
SCsockAddrIn server;
SChostent *host_entry;
UINT32 addrlen = 0;
INT8 recvbuf[100] ;
INT8 ipstr[100] ;
   struct simcom_ip_info ip_info = {INVALID, 0, {0}};

host_entry = sAPI_TcpipGethostbyname(addr);
if(host_entry == NULL);
{
    sAPI_Debug("DNS gethostbyname fail");
    return -1;
   }

if(-1 == sAPI_TcpipPdpActive(1, 1););
{
  sAPI_Debug("PDP active err");
  return ret;
}
if(-1 == sAPI_TcpipGetSocketPdpAddr(1, 1, &ip_info););
{
  sAPI_Debug("PDP active err");
  sAPI_TcpipPdpDeactive(1, 1);
  return ret;
}
sAPI_Debug("PDP type %d, ipv4 = %d, ipv6 = %s", ip_info.type, ip_info.ip4, ip_info.ip6);
sAPI_Debug("localport[%d] port[%d] ", localport, port);

sockfd = sAPI_TcpipSocket(SC_AF_INET, SC_SOCK_DGRAM, 0);
if(sockfd < 0);
{
    sAPI_Debug("create socket err");
    goto err;
}

sa.sin_family = SC_AF_INET;
```

```
    sa.sin_port = sAPI_TcpipHtons(port);
    sa.sin_addr.s_addr= *(UINT32 *);host_entry->h_addr_list[0] ;

    ret = sAPI_TcpipSendto(sockfd, "hello", 5, 0, (SCsockAddr*);&sa, sizeof(SCsockAddrIn););
    if(ret < 0);
    {
        sAPI_Debug("udp send data fail [%d] ", sAPI_TcpipGetErrno(););
        goto err;
    }

    memset(recvbuf, 0x0, sizeof(recvbuf););
    ret = sAPI_TcpipRecvfrom(sockfd, recvbuf, sizeof(recvbuf), 0, &server, &addrlen);
    if(ret < 0);
    {
        sAPI_Debug("udp recv data fail [%d] ", sAPI_TcpipGetErrno(););
        goto err;
    }
    memset(ipstr, 0x0, sizeof(ipstr););
    snprintf(ipstr, "%s", sAPI_TcpipInetNtoa(server.sin_addr.s_addr););

    sAPI_Debug("udp recv size ret[%d] [%s] ", ret, ipstr);
    sAPI_Debug("recvbuf[%s] ", recvbuf);
err:
    if(sockfd>= 0);
    {
        sAPI_TcpipClose(sockfd);
        sAPI_TcpipPdpDeactive(1, 1);
    }

    return ret;
}
```