移植sdk-c说明文档

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订日期** | **修订内容** | **修订人** |
| V0.1 | 2016.01.28 | 新增 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Sdk-c目录结构简介

sdk-c

├ include // regina sdk API相关头文件

│ └ regina

│ └ platform.h // 系统相关的整数类型定义

├ platforms

│ ├ include // 系统相关的定义文件

│ └ src

│ ├ posix // posix系统的实现文件

│ ├ win // Windows系统的实现文件

├ src // sdk-c实现的源代码

└ thirdparty

上面的目录结构中，红色字体是开发支持新系统相关部分。下面对这两个目录下面的文件做一个说明

–– platform.h // 定义了sdk所需的size\_t, int16\_t, int32\_t, int64\_t, uint16\_t, uint32\_t, uint64\_t等整数类型。

–– platforms/include

├ environment.h // 初始化平台相关环境接口

├ std\_wrapper.h // 标准C库函数的包装

├ tcp\_util.h // tcp socket相关的定义接口

└ thread.h // 线程相关的定义接口

–– platforms/src

├ posix

│ ├ allocator.c // posix系统环境内存分配相关函数实现

│ ├ environment.c // posix系统环境初始化实现

│ ├ std\_wrapper.c // posix系统的C库包装函数实现

│ ├ tcp\_util.c // posix系统的tcp socket相关实现

│ └ thread.c // posix系统的线程相关实现

├ win

│ ├ allocator.c // Windows系统环境内存分配相关函数实现

│ ├ environment.c // Windows系统环境初始化实现

│ ├ std\_wrapper.c // Windows系统的C库包装函数实现

│ ├ tcp\_util.c // Windows系统的tcp socket相关实现

│ └ thread.c // Windows系统的线程相关实现

第2章以实现支持windows系统为例，对开发支持新系统做一个说明。

1. 移植说明
   1. 开发步骤说明
2. 在sdk-c/platforms/src/目录下，创建win目录
3. 在sdk-c/platforms/src/win/目录下，创建实现文件（可以仿照posix目录下的文件结构）。
4. 实现sdk-c/platforms/srcwin/目录下的文件

* allocator.c
* environment.c
* std\_wrapper.c
* tcp\_util.c
* thread.c

具体的实现可以参见第2.2节以及posix的实现文件

* 1. 定义类型和实现接口

1. 初始化系统环境的实现

* 类型
* 函数
* int initialize\_environment( void );
* void cleanup\_environment( void );

由于Windows的Socket库需要初始化，所以可以在initialize\_environment的实现中做socket库的初始化操作，在cleanup\_environment中做socket库的清理工作：

int initialize\_environment( void )

{

WSADATA wsa;

int err = 0;

err = WSAStartup( MAKEWORD(2,2), &wsa );

if (0 != err)

{

return -1;

}

return 0;

}

void cleanup\_environment( void )

{

WSACleanup();

}

1. TCP Socket相关实现(tcp\_util.c)

* 类型
* socket\_fd\_t

struct socket\_fd\_t {

SOCKET fd;

};

* ip\_address\_t

struct ip\_address\_t {

int len;

struct sockaddr \*addr;

};

* 函数
* void tcp\_util\_ip\_address\_destroy(ip\_address\_t \*address)
* int tcp\_util\_getaddrinfo(const char \*host, unsigned int port, int ipv6,

ip\_address\_t \*\*addresses, unsigned int \*addrcnt);

* int tcp\_util\_open(socket\_fd\_t \*\*fd, ip\_address\_t \*address, unsigned int timeout);
* int tcp\_util\_send(socket\_fd\_t \*fd, const char \*buf,

unsigned int size, unsigned int \*bytes\_sent, int flags);

* int tcp\_util\_recv(socket\_fd\_t \*fd, char \*buf,

unsigned int size, unsigned int \*bytes\_read, int flags);

* int tcp\_util\_close(socket\_fd\_t \*fd);

注：所有的tcp通信均为**非阻塞模式**。

1. 线程实现 (thread.h,thread.c)

* 函数
* int regina\_sleep(int milliseconds);

例如：windows下有thread和mutex相对于的实现机制，修改如下：

usleep 🡺 Sleep