socket编程

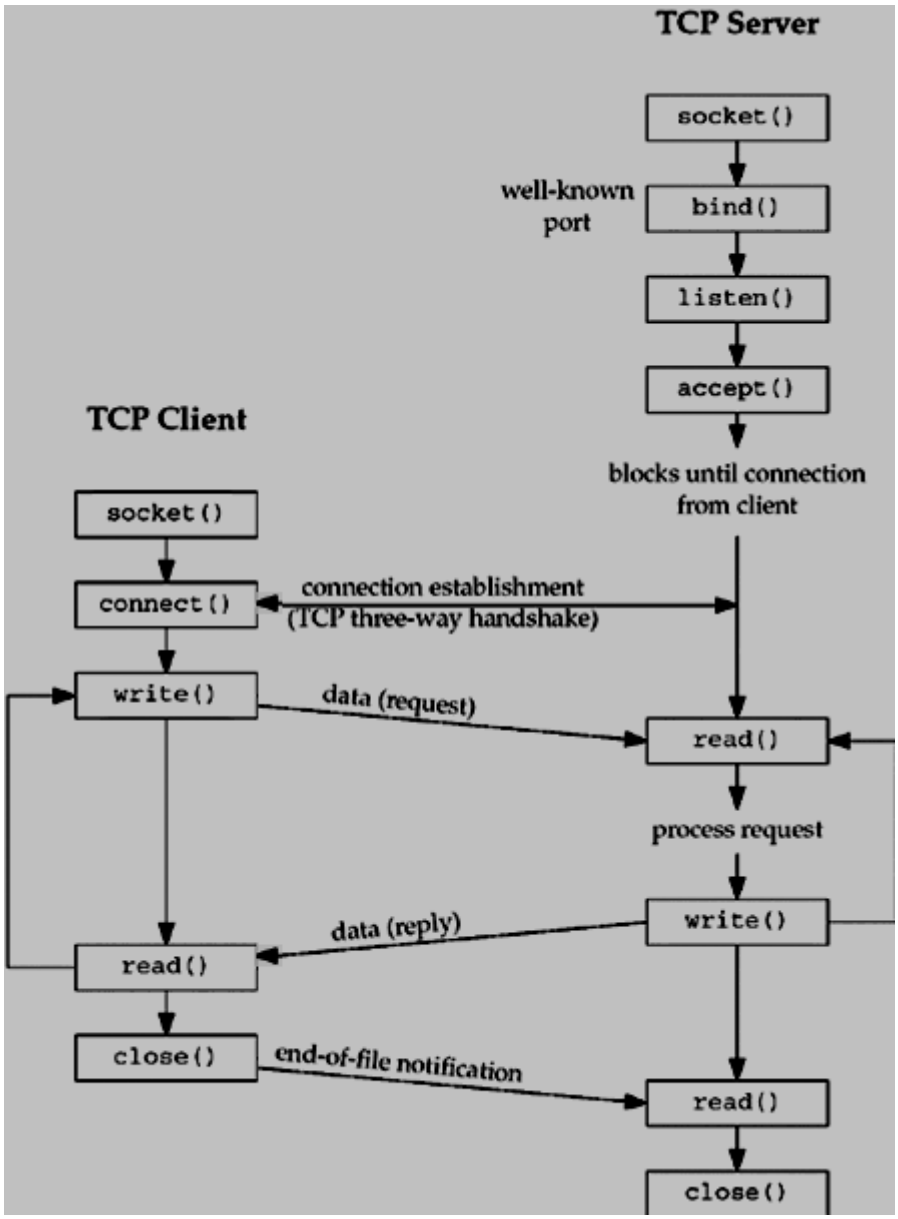
1. socket 学习

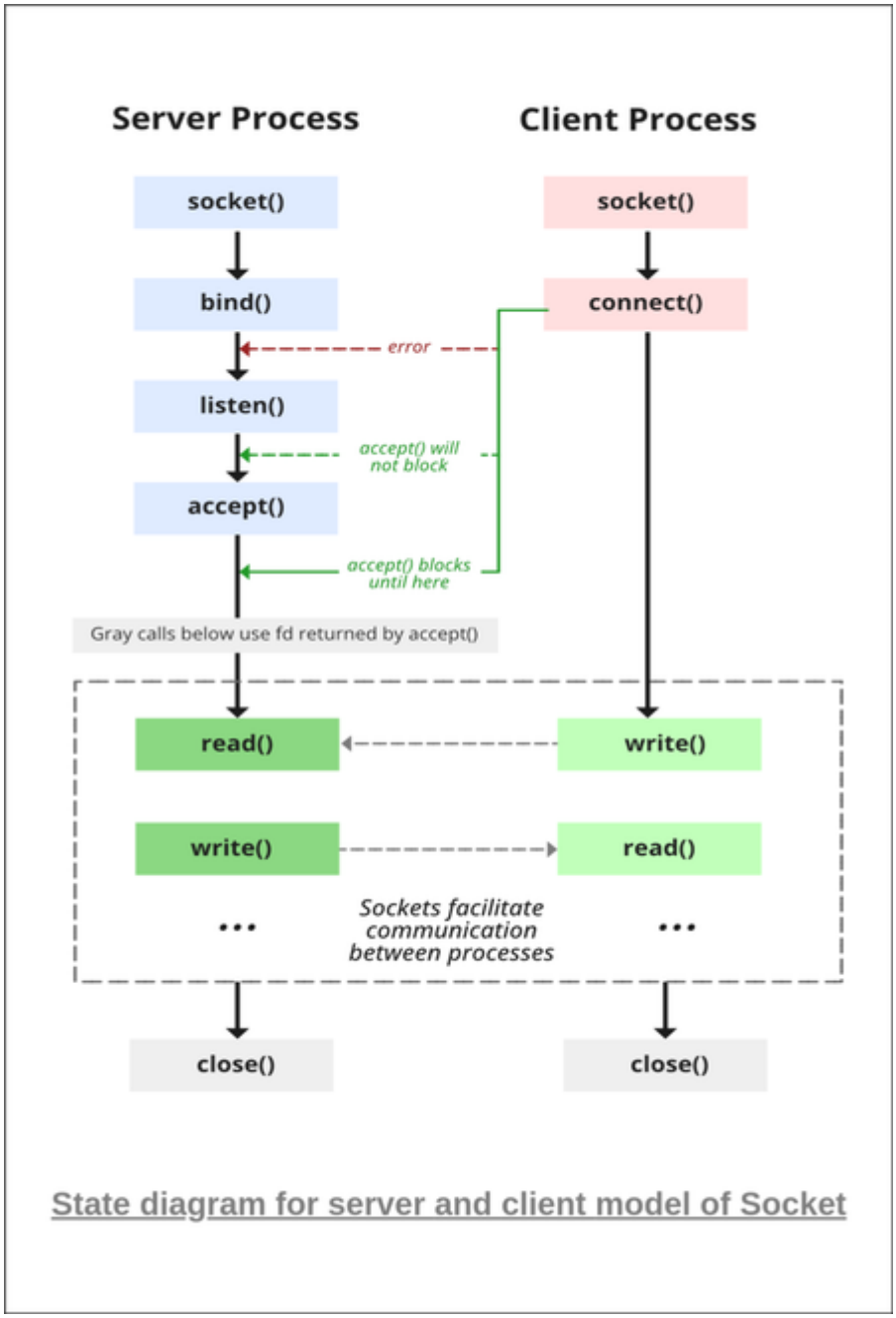
Unix Socket - Core Functions

Socket Programming in C/C++

Socket programming in C on Linux – The Ultimate Guide for Beginners

1.1 大图





State diagram for server and client model of Socket

1.2 function

```
1. Socket creation:
int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)
domain: integer, specifies communication domain. We use AF_ LOCAL as defined
in the POSIX standard for communication between processes on the same host.
For communicating between processes on different hosts connected by IPV4, we
use AF_INET and AF_I NET 6 for processes connected by IPV6.
type: communication type
SOCK_STREAM: TCP(reliable, connection oriented)
SOCK_DGRAM: UDP(unreliable, connectionless)
protocol: Protocol value for Internet Protocol(IP), which is 0. This is the
same number which appears on protocol field in the IP header of a
```

```
packet.(man protocols for more details)
2. Setsockopt:
This helps in manipulating options for the socket referred by the file
descriptor sockfd. This is completely optional, but it helps in reuse of
address and port. Prevents error such as: “address already in use”.

int setsockopt(int sockfd, int level, int optname,  const void *optval,
socklen_t optlen);

3. Bind:
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);


After the creation of the socket, the bind function binds the socket to the
address and port number specified in addr(custom data structure). In the
example code, we bind the server to the localhost, hence we use INADDR_ANY
to specify the IP address.

4. Listen:
int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client
to approach the server to make a connection. The backlog, defines the
maximum length to which the queue of pending connections for sockfd may
grow. If a connection request arrives when the queue is full, the client may
receive an error with an indication of ECONNREFUSED.

5. Accept:
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t
*addrlen);

It extracts the first connection request on the queue of pending connections
for the listening socket, sockfd, creates a new connected socket, and
returns a new file descriptor referring to that socket. At this point, the
connection is established between client and server, and they are ready to
transfer data.

Stages for Client
Socket connection: Exactly same as that of server’s socket creation
Connect: The connect() system call connects the socket referred to by the
file descriptor sockfd to the address specified by addr. Server’s address
and port is specified in addr.
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Implementation
Here we are exchanging one hello message between server and client to
demonstrate the client/server model.

Server.c
// Server side C/C++ program to demonstrate Socket
// programming
```

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = { 0 };
    char* hello = "Hello from server";
    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET,
                   SO_REUSEADDR | SO_REUSEPORT, &opt,
                   sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr*)&address,
             sizeof(address))
        < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket
         = accept(server_fd, (struct sockaddr*)&address,
                  (socklen_t*)&addrlen))
        < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read(new_socket, buffer, 1024);
    printf("%s\n", buffer);
    send(new_socket, hello, strlen(hello), 0);
```

```
    printf("Hello message sent\n");
    // closing the connected socket
    close(new_socket);
    // closing the listening socket
    shutdown(server_fd, SHUT_RDWR);
    return 0;
}
client.c
// Client side C/C++ program to demonstrate Socket
// programming
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
    int status, valread, client_fd;
    struct sockaddr_in serv_addr;
    char* hello = "Hello from client";
    char buffer[1024] = { 0 };
    if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        return -1;
    }
    if ((status
         = connect(client_fd, (struct sockaddr*)&serv_addr,
                   sizeof(serv_addr)))
        < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
    send(client_fd, hello, strlen(hello), 0);
    printf("Hello message sent\n");
    valread = read(client_fd, buffer, 1024);
    printf("%s\n", buffer);
    // closing the connected socket
    close(client_fd);
    return 0;
}
```

```
Compiling:

gcc client.c -o client
gcc server.c -o server
Output:

Client:Hello message sent
Hello from server
Server:Hello from client
Hello message sent
Next: Socket Programming in C/C++: Handling multiple clients on server
without multi threading


This article is contributed by Akshat Sinha. If you like GeeksforGeeks and
would like to contribute, you can also write an article using
write.geeksforgeeks.org or mail your article to review-
team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main
page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share
more information about the topic discussed above.
```

2. 示例代码

参考：https://blog.csdn.net/liranke/article/details/6079393/

这个代码在linux系统下可直接使用，同样也适于xilinx zynq petalinx下使用。

2.1 服务端代码

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_READ_LINE 1024

int main(void) {
    int recv_len = -1;
    int conn_fd = -1;
    int ret = -1;
```

```
    //ip端口，必须和client的一样，才能正常建立链接
    int server_ip_port = 996;

    //用于存储接收到的数据
    char buff[MAX_READ_LINE];

    //初始化sockaddr_in结构体
    struct sockaddr_in t_sockaddr;
    memset(&t_sockaddr, 0, sizeof(t_sockaddr));
    t_sockaddr.sin_family = AF_INET;
    t_sockaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    t_sockaddr.sin_port = htons(server_ip_port);

    //创建server端的socket套接字
    int listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd < 0) {
        fprintf(stderr, "socket error %s errno: %d\n", strerror(errno),
errno);
    }

    //绑定
    ret = bind(listen_fd,(struct sockaddr *)
&t_sockaddr,sizeof(t_sockaddr));
    if (ret < 0) {
        fprintf(stderr, "bind socket error %s errno: %d\n", strerror(errno),
errno);
    }

    //监听
    ret = listen(listen_fd, 1024);
    if (ret < 0) {
        fprintf(stderr, "listen error %s errno: %d\n", strerror(errno),
errno);
    }

    //接收来自客户端的链接请求
    for(;;) {
        conn_fd = accept(listen_fd, (struct sockaddr*)NULL, NULL);
        if(conn_fd < 0) {
            fprintf(stderr, "accpet socket error: %s errno :%d\n",
strerror(errno), errno);
            continue;
        }

        //读取数据到buff中
        recv_len = recv(conn_fd, buff, MAX_READ_LINE, 0);
        if (recv_len < 0) {
            fprintf(stderr, "recv error %s errno: %d\n", strerror(errno),
errno);
            continue;
```

```
        }

        buff[recv_len] = '\0';
        //将接收到的数据标准输出
        fprintf(stdout, "recv msg from client: %s\n", buff);
        close(conn_fd);
        conn_fd = -1;
    }

    //关闭套接字
    close(listen_fd);
    listen_fd = -1;

    return 0;
}
```

2.2 客户端代码

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main(void) {
    char *server_ip_addr = "127.0.0.1";
    int server_ip_port = 996;

    //要发送给server的数据
    char *send_message = "hello";

    //创建client端的socket套接口
    int socket_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_fd < 0) {
        fprintf(stderr, "socket error %s errno: %d\n", strerror(errno),
errno);
    }

    //初始化sockaddr_in结构体
    struct sockaddr_in t_sockaddr;
    memset(&t_sockaddr, 0, sizeof(struct sockaddr_in));
    t_sockaddr.sin_family = AF_INET;
    t_sockaddr.sin_port = htons(server_ip_port);
    inet_pton(AF_INET, server_ip_addr, &t_sockaddr.sin_addr);

    //连接
    if((connect(socket_fd, (struct sockaddr*)&t_sockaddr, sizeof(struct
```

```
sockaddr))) < 0 ) {
        fprintf(stderr, "connect error %s errno: %d\n", strerror(errno),
errno);
        return 0;
    }

    //向server发送数据
    if((send(socket_fd, send_message, strlen(send_message), 0)) < 0) {
        fprintf(stderr, "send message error: %s errno : %d",
strerror(errno), errno);
        return 0;
    }

    //关闭套接字
    close(socket_fd);
    socket_fd = -1;

    return 0;
}
```