学号:031602523姓名:刘宏岩 学院:数计学院 专业:计算机类 班级:实验班

**《Linux 操作系统设计实践》**

**实验三:网络编程**

* **实验环境:Ubuntu16.04**
* **实验内容:**

1. **套接字socket：**

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。Socket的英文原义是“孔”或“插座”。作为BSD UNIX的进程通信机制，取后一种意思。通常也称作"套接字"，用于描述**IP地址和端口**，是一个通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。在Internet上的主机一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务。Socket正如其英文原意那样，像一个多孔插座。一台主机犹如布满各种插座的房间，每个插座有一个编号， 客户软件将插头插到不同编号的插座，就可以得到不同的服务。

1. **相关API学习**

* socket的接口函数声明在头文件sys/types.h和sys/socket.h中

1. **socket——创建套接字**

该函数用来创建一个套接字，并返回一个描述符，该描述符可以用来访问该套接字，它的原型如下：

int socket(int domain, int type, int protocol);

函数中的三个参数分别对应前面所说的三个套接字属性。protocol参数设置为0表示使用默认协议

1. **bind——命名（绑定）套接字**

该函数把通过socket调用创建的套接字命名，从而让它可以被其他进程使用。对于AF\_UNIX，调用该函数后套接字就会关联到一个文件系统路径名，对于AF\_INET，则会关联到一个IP端口号。函数原型如下：

int bind( int socket, const struct sockaddr \*address, size\_t address\_len);

成功时返回0，失败时返回-1；

1. **Sendto——发送数据**

该函数把缓冲区buffer中的信息给送给指定的IP端口的程序，原型如下：

int sendto(int sockfd, void \*buffer, size\_t len, int flags, struct sockaddr \*to, socklen\_t tolen);

buffer中储存着将要发送的数据，len是buffer的长度，而flags在应用中通常被设置为0，to是要发送数据到的程序的IP端口，tolen是to参数的长度。

成功时返回发送的数据的字节数，失败时返回-1.

1. **Recvfrom——接收数据**

该函数把发送给程序的信息储存在缓冲区buffer中，并记录数据来源的程序IP端口，原型如下：

int recvfrom(int sockfd, void \*buffer, size\_t len，int flags, struct sockaddr \*src\_from, socklen\_t \*src\_len);

buffer用于储存接收到的数据，len指定buffer的长度，而flags在应用中通常被设置0，src\_from若不为空，则记录数据来源程序的IP端口，若src\_len不为空，则其长度信息记录在src\_len所指向的变量中。

**注意：默认情况下，recvfrom是一个阻塞的调用，即直到它接收到数据才会返回。**

1. **Close——关闭socket**

该系统调用用来终止服务器和客户上的套接字连接，我们应该总是在连接的两端（服务器和客户）关闭套接字。

1. **服务器端**

* 服务器应用程序用系统调用socket来创建一个套接口，它是系统分配给该服务器进程的类似文件描述符的资源，它**不能与其他的进程共享。**
* 接下来，服务器进程会给套接字起个名字（监听），我们使用系统调用bind来给套接字命名。然后服务器进程就开始等待客户连接到这个套接字。
* 然后系统调用recvfrom来接收来自客户程序发送过来的数据。服务器程序对数据进行相应的处理，再通过系统调用sendto把处理后的数据发送回客户程序。

1. **客户端**

* 基于数据报socket的客户端比服务器端简单，同样，客户应用程序首先调用socket来创建一个未命名的套接字，与服务器一样，客户也是通过sendto和recvfrom来向服务器发送数据和从服务器程序接收数据。
* **实验总结:**

1. **实验流程图**

****

1. **服务器端代码**

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

int main()

{

int server\_sockfd = -1;

int server\_len = 0;

int client\_len = 0;

char buffer[512];

int result = 0;

struct sockaddr\_in server\_addr;

struct sockaddr\_in client\_addr;

//创建数据报套接字

server\_sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

//设置监听IP端口

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(9739);

server\_len = sizeof(server\_addr);

//绑定（命名）套接字

bind(server\_sockfd, (struct sockaddr\*)&server\_addr, server\_len);

//忽略子进程停止或退出信号

signal(SIGCHLD, SIG\_IGN);

while(1)

{

//接收数据，用client\_addr来储存数据来源程序的IP端口

result = recvfrom(server\_sockfd, buffer, sizeof(buffer), 0,

(struct sockaddr\*)&client\_addr, &client\_len);

if(fork() == 0)

{

//利用子进程来处理数据

buffer[0] += 'a' - 'A';

sleep(5);

//发送处理后的数据

sendto(server\_sockfd, buffer, sizeof(buffer),0 ,

(struct sockaddr\*)&client\_addr, client\_len);

printf("%c\n", buffer[0]);

//注意，一定要关闭子进程，否则程序运行会不正常

exit(0);

}

}

//关闭套接字

close(server\_sockfd);

}

1. **客户端代码**

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <stdio.h>

int main(int agrc, char \*argv[])

{

struct sockaddr\_in server\_addr;

int server\_len = 0;

int sockfd = -1;

int result = 0;

char c = 'A';

//取第一个参数的第一个字符

if(agrc > 1)

c = argv[1][0];

//创建数据报套接字

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

//设置服务器IP端口

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(9739);

server\_len = sizeof(server\_addr);

//向服务器发送数据

sendto(sockfd, &c, sizeof(char), 0,

(struct sockaddr\*)&server\_addr, server\_len);

//接收服务器处理后发送过来的数据，由于不关心数据来源，所以把后两个参数设为0

recvfrom(sockfd, &c, sizeof(char), 0, 0, 0);

printf("char from server = %c\n", c);

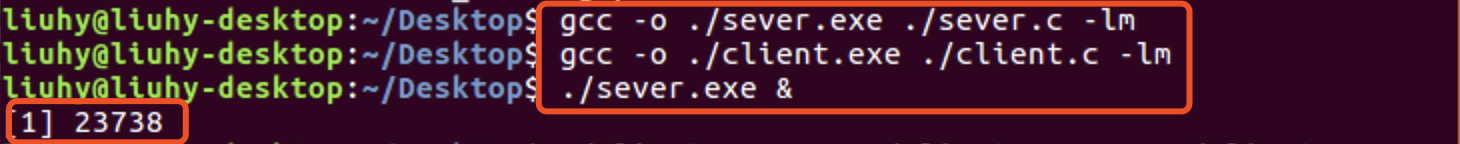
//关闭套接字

close(sockfd);

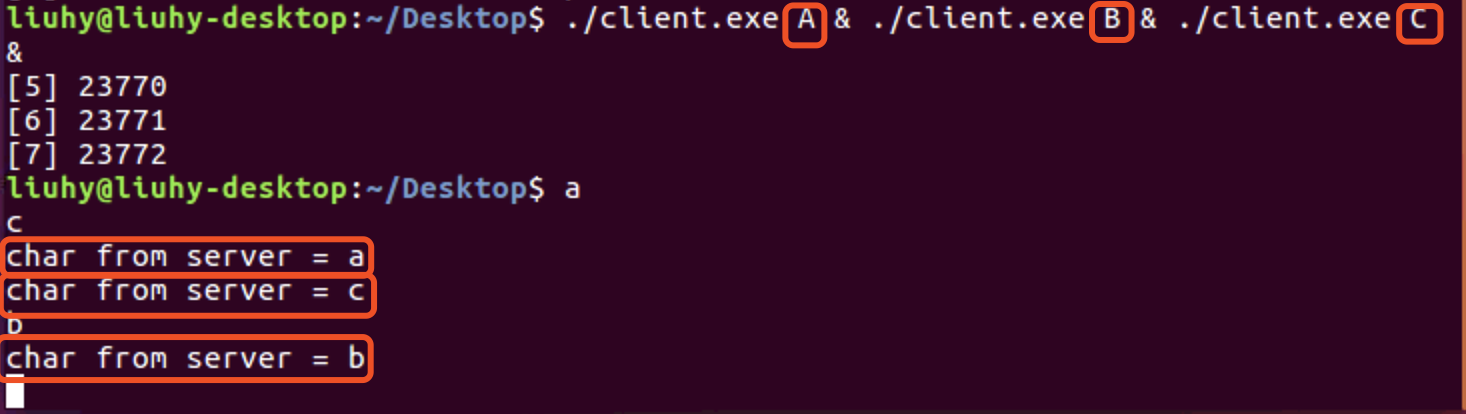
exit(0);

}

1. **实验截图**
2. **运行服务器**

****

1. **客户端发送数据，大写转化为小写**

****