实验一_实验报告

题目：秒杀模拟器设计与实现	姓名：干艳桃	学号：2012103315

一、实验目的

1. 熟悉客户/服务器编程模式
2. 熟悉TCP、UDP协议原理
3. 熟悉Socket编程方法
4. 巩固UNIX/LINUX环境下的system call

二、实验平台

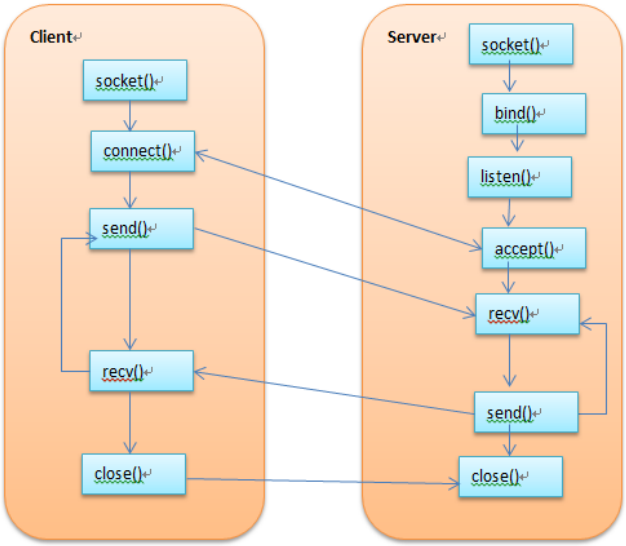
1. OS: Ubuntu-10.04 (Kernel：2.6.32-21-server)
2. compiler: gcc 4.4.3

三、设计和原理

1. **C/S编程**：服务端有一个进程（或多个进程）在指定的端口等待客户来连接，服务程序等待客户的连接信息，一旦连接上之后，就可以按设计的数据交换方法和格式进行数据传输。客户端在需要的时刻发出向服务端的连接请求。客户端程序，主要指发出服务请求的程序。客户端提供用户界面（UI或GUI），完成用户的输入，实现某些简单的处理，向服务器提交请求，并对服务器送回的信息进行分析，提交给用户。服务器向客户机提供一种服务，服务的的类型由系统定；服务器只负责响应来自客户机的查询或命令，不主动提供联系。服务器与客户进行通信时，客户端程序设计解决的问题包括：服务端的地址，服务所提供的端口号，服务使用UDP还是TCP协议。服务端程序：侦听某个端口，等待请求消息的到来。如果请求消息被认可，则返回应答消息。服务端程序分为：基于TCP协议和UDP协议的两种方式，从结构上可划分为：循环和并发。

2. **TCP、UDP协议**：TCP（Transmission Control Protocol，传输控制协议）是一种面向连接的、可靠的运输层协议，为用户提供可靠的、全双工的字节流服务。TCP是可靠的，并且使用确认机制。UDP（User DataGram Protocl，用户数据报协议）是一种简单的面向数据报的传输层协议。是一种无连接协议，不保证数据报一定能到达目的地，不具有可靠性，不支持确认和重发。

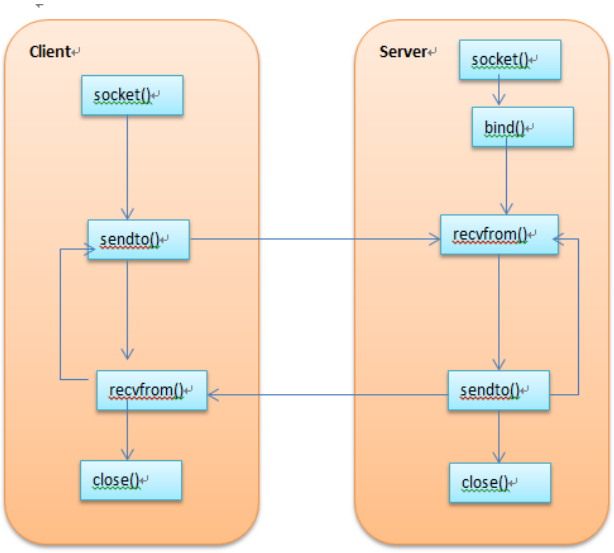
3. **基于TCP协议的C/S编程**：



服务端进程：1）使用socket调用得到一个描述符；2）使用bind调用将一个名字与socket描述符连接起来，对于Internet域就是将Internet地址联编到socket；3）服务端使用listen调用指出等待服务请求队列的长度；4)使用accept调用等待客户端发起连接，一旦有客户端发出连接，accept返回客户的地址信息，并返回一个新的socket描述符，该描述符与原先的socket有相同的特性；5）服务端使用新的socket进行读写操作，一般服务端可能在accept返回后创建一个新的进程进行与客户的通信，父进程则再到accept调用处等待另一个连接。

客户端进程：1）使用socket调用得到一个socket描述符；2）使用connet向指定的服务器上的指定端口发起连接，一旦连接成功返回，就说明已经建立了与服务器的连接；3）通过socket描述符进行读写操作。

4. **基于UDP协议的C/S编程：**



服务端进程：1）使用socket调用得到一个描述符；2）使用bind调用将一个名字与socket描述符连接起来，对于Internet域就是将Internet地址联编到socket；3）调用recvfrom等待来自客户的数据到达，接收客户端的数据报，此系统调用返回客户进程的网络地址和数据报，服务器可向对方进程发出响应；4）调用sendto将要返回客户端的消息发送给客户进程。

客户端：1）创建一个socket；2）使用sendto向服务端进程发一个数据报，作为参数请求目的地址（服务器端）；3）使用recvfrom得到返回的消息。

5. **Socket**：即"套接字"，用于描述IP地址和端口，是一个通信链的句柄。应用程序通常通过"套接字"向网络发出请求或者应答网络请求。

四、实验内容

1. **程序功能**：受淘宝双十一活动启发，编写一个秒杀模拟器。由客户端发起秒杀请求，服务器端返回问题（模拟验证码），用户回答问题并将答案送回服务器端，如果答案正确且秒杀商品还未被秒走，则告知客户端秒杀成功。

2. **实验假设**：本实验共使用2台机器，其中1台为服务器，1台为客户端，用两个客户端程序来模拟2个客户。并假定秒杀商品只有1件，2个用户竞争秒杀。采用基于TCP协议的C/S编程。

3. **相关system call：**

①socket()：

声明格式：int socket(int domain, int type, int protocol);

调用时加头文件：#include<sys/socket.h>

功能：创建一个套接字，调用成功返回套接字描述符，出错返回-1.

②bind()：

声明格式：int bind(int sockfd, const struct sockaddr *addr, socket_t len);
调用时加头文件：#include<sys/socket.h>
功能：将地址绑定到一个套接字，调用成功返回0，出错返回-1.

③ listen()：

声明格式：int listen(int sockfd, int backlog);
调用时加头文件：#include<sys/socket.h>
功能：宣告可以接受连接请求，调用成功返回0，出错返回-1.

④connect()：

声明格式：int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
调用时加头文件：#include<sys/socket.h>
功能：建立一个连接，调用成功返回0，出错返回-1.

⑤ accept()：

声明格式：int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict len);
调用时加头文件：#include<sys/socket.h>
功能：获得连接请求并建立连接，调用成功返回套接字描述符，出错返回-1.

⑥ send ()：

声明格式：asize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
调用时加头文件：#include<sys/socket.h>
功能：用来发送数据，类似于write，调用成功返回发送的字节数，出错返回-1.

⑦ recv()：

声明格式：asize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
调用时加头文件：#include<sys/socket.h>
功能：用来接收数据，类似于read，调用成功返回接收的字节数，出错返回-1.

⑧ close()：

声明格式：int close(int sockfd);
调用时加头文件：#include<sys/socket.h>
功能：终止连接，调用成功返回0，出错返回-1.

五、程序清单

1. 客户端程序：

```
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <netdb.h>
#include <arpa/inet.h>
#define HELLO_WORLD_SERVER_PORT       6666
#define BUFFER_SIZE 1024
#define FILE_NAME_MAX_SIZE 512
int main(int argc, char **argv){
    if (argc != 2){
        printf("Usage: ./%s ServerIPAddress\n",argv[0]);//Usage：./client ServerIPAddress
        exit(1);}
    //设置一个socket地址结构client_addr，代表客户机internet地址，端口
    struct sockaddr_in client_addr;
    bzero(&client_addr,sizeof(client_addr)); // sin_zero为填充字段，须置为0，故赋值前这个地址数据结构中的内容全部置0
    client_addr.sin_family = AF_INET;       // AF_NET：IPv4因特网域
    client_addr.sin_addr.s_addr = htons(INADDR_ANY);//INADDR_ANY表示自动获取本机地址
    client_addr.sin_port = htons(0);       //0表示让系统自动分配一个空闲端口
    //创建用于TCP协议的socket
    int client_socket = socket(AF_INET,SOCK_STREAM,0);
    if( client_socket < 0){
        printf("Create Socket Failed!\n");
        exit(1);}
    //设置一个socket地址结构server_addr，代表服务器的internet地址，端口
    struct sockaddr_in server_addr;
    bzero(&server_addr,sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    if(inet_aton(argv[1],&server_addr.sin_addr) == 0) //服务器的IP地址来自第一个输入参数      {
        printf("Server IP Address Error!\n");
        exit(1);}
    server_addr.sin_port = htons(HELLO_WORLD_SERVER_PORT);
    socklen_t server_addr_length = sizeof(server_addr);
    //向服务器发起连接
    if(connect(client_socket,(struct sockaddr*)&server_addr, server_addr_length) < 0){
```

```
            printf("Can Not Connect To %s!\n",argv[1]);
            exit(1);}
        char ans[100];
        int length=0;
        char name[FILE_NAME_MAX_SIZE+1];
        bzero(name, FILE_NAME_MAX_SIZE+1);
        printf("Please Input your name: ");
        scanf("%s",name);
        //连接成功后，让用户输入用户名，模拟淘宝登录
        send(client_socket,name,strlen(name),0);
        char buffer[BUFFER_SIZE];
        bzero(buffer,BUFFER_SIZE);
        //接收验证问题，模拟验证码
        length = recv(client_socket,buffer,BUFFER_SIZE,0);
        printf("%s",buffer);
        if(strcmp(buffer,"Seckill is over!")==0){ //如果服务端回复的消息为“秒杀结束”，则关闭socket连接
    close(client_socket);
            return 0;}
        bzero(ans,100);
        scanf("%s",ans);
        //发送答案给服务器
        send(client_socket,ans,strlen(ans),0);
        bzero(buffer,BUFFER_SIZE);
        length= recv(client_socket,buffer,BUFFER_SIZE,0);
        //秒杀成功
        while(strcmp(buffer,"right")){
            printf("%s",buffer);
            bzero(ans,100);
            scanf("%s",ans);
            send(client_socket,ans,strlen(ans),0);
            bzero(buffer,BUFFER_SIZE);
            length= recv(client_socket,buffer,BUFFER_SIZE,0);
        }
        printf("Answer is right\n");
        bzero(buffer,BUFFER_SIZE);
        length= recv(client_socket,buffer,BUFFER_SIZE,0);
        printf("%s",buffer);
        close(client_socket);
        return 0;
}
```

1. 服务器端程序：

```
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/signal.h>
#include <unistd.h>
#define HELLO_WORLD_SERVER_PORT       6666 //使用的端口号
#define LENGTH_OF_LISTEN_QUEUE 20
#define BUFFER_SIZE 1024
#define FILE_NAME_MAX_SIZE 512
void reaper(int sig){
    int status;
    //调用wait3读取子进程的返回值，使僵死状态的子进程彻底释放
     while(wait3(&status,WNOHANG,(struct rusage*)0) >=0);
}
int main(int argc, char **argv){
    //设置一个socket地址结构server_addr，代表服务器internet地址，端口
    struct sockaddr_in server_addr;
    bzero(&server_addr,sizeof(server_addr)); //把一段内存区的内容全部设置为0
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htons(INADDR_ANY);
    server_addr.sin_port = htons(HELLO_WORLD_SERVER_PORT);
    //创建基于TCP协议的socket
    int server_socket = socket(PF_INET,SOCK_STREAM,0);
    if( server_socket < 0){
        printf("Create Socket Failed!");
        exit(1);
    }
    {
        int opt =1;
        setsockopt(server_socket,SOL_SOCKET,SO_REUSEADDR,&opt,sizeof(opt));
    }
    //把socket和socket地址结构联系起来
    if( bind(server_socket,(struct sockaddr*)&server_addr,sizeof(server_addr))){
        printf("Server Bind Port : %d Failed!", HELLO_WORLD_SERVER_PORT);
        exit(1);}
```

```
    //监听客户端的请求
    if ( listen(server_socket, LENGTH_OF_LISTEN_QUEUE) ){
        printf("Server Listen Failed!");
        exit(1);}
   //通知OS，当收到子进程的退出信号时，执行reaper函数，释放僵死状态的进程
    (void)signal(SIGCHLD,reaper);
    while (1) //服务器端一直运行着      {
        //定义客户端的socket地址结构client_addr
        struct sockaddr_in client_addr;
        socklen_t length = sizeof(client_addr);
        //接收一个到server_socket代表的socket的一个连接
        //如果没有连接请求，就等待到有连接请求—这是accept函数的特性
        //accept函数返回一个新的socket，这个socket用于同连接到的客户通信
        int new_server_socket = accept(server_socket,(struct sockaddr*)&client_addr,&length);
        if ( new_server_socket < 0){
            printf("Server Accept Failed!\n");
            break;}
        int child_process_pid = fork(); //fork()后，子进程是主进程的拷贝
        if(child_process_pid==0){//子进程执行与客户端的交互
        // close(server_socket);
        int tag;
        char buffer[BUFFER_SIZE];
        bzero(buffer,BUFFER_SIZE);
        length = recv(new_server_socket,buffer,BUFFER_SIZE,0);//接收客户端的用户名
        if (length < 0){
            printf("Server Recieve The User Name Failed!\n");
            break;
        }
char name[FILE_NAME_MAX_SIZE+1];
bzero(name, FILE_NAME_MAX_SIZE+1);
strncpy(name, buffer, strlen(buffer)>FILE_NAME_MAX_SIZE?FILE_NAME_MAX_SIZE:strlen(buffer));
printf("%s is requesting\n",name);

FILE* fp = fopen("tag.txt","r+");
if(NULL == fp ){
        printf("File:tag.txt Can Not Open To Read\n");
        exit(1);
       }
fscanf(fp,"%d",&tag);
bzero(buffer, BUFFER_SIZE);
if(tag==1){//用tag来标记秒杀成功的次数
    strcpy(buffer,"Seckill is over!\n");
    printf("Seckill is over!\n");
    length = send(new_server_socket,buffer,sizeof(buffer),0);//发送“秒杀结束”给客户端
```

```
            fclose(fp);
        }else{
            strcpy(buffer,"Please answer the question: 3+5=?\nInput the answer: ");
            printf("send question\n");
            length = send(new_server_socket,buffer,sizeof(buffer),0);//发送验证问题给客户端
            bzero(buffer, BUFFER_SIZE);
            length = recv(new_server_socket,buffer,BUFFER_SIZE,0); //接收客户端的答案
            while(strcmp(buffer,"8")){
                        printf("Answer %s is wrong\n",buffer);
                        bzero(buffer,BUFFER_SIZE);
                        strcpy(buffer,"Answer is wrong!,continue\nInput the answer: ");
                        send(new_server_socket,buffer,sizeof(buffer),0); //告诉用户答案错误
                        bzero(buffer,BUFFER_SIZE);
                        length = recv(new_server_socket,buffer,BUFFER_SIZE,0);
                    }
                    printf("Answer %s is right\n",buffer);
                    bzero(buffer,BUFFER_SIZE);
                    strcpy(buffer,"right");
                    send(new_server_socket,buffer,sizeof(buffer),0); //告诉用户答案正确

                    fseek(fp,0,SEEK_SET);
                    fscanf(fp,"%d",&tag);
                    bzero(buffer,BUFFER_SIZE);
                    if(tag==1){
                      strcpy(buffer,"Seckill is over!\n");
                      printf("Seckill is over!\n");
                      send(new_server_socket,buffer,sizeof(buffer),0);//秒杀结束
                    }else{
                    tag=1;
                    fseek(fp,0,SEEK_SET);
                    fprintf(fp,"%d",tag);
                    strcpy(buffer,"Congratulation,Seckill is successful!\n");//秒杀成功
                    send(new_server_socket,buffer,sizeof(buffer),0);
                    }
                    fclose(fp);
                }
            }else if(child_process_pid > 0) //父进程
              close(new_server_socket);     //父进程中不需要用于客户端交互的new_server_socket
        }
        //关闭监听用的socket
        close(server_socket);
        return 0;
    }
```

六、实验结果

1. 在机器wamdm52（IP为202.112.113.252）上启动服务器，开始监听客户端请求：

```
wamdm@WAMDM52:~/taozi/ad_OS/socketnew$ ./server
server start...
```

2. 在机器wamdm51上启动2个客户端程序，向服务器发起连接请求，连接成功后，服务器要求用户输入用户名：

```
wamdm@WAMDM51:~/taozinew/socketnew$ ./client 202.112.113.252
Please Input your name:
```

3. 分别在2个客户端程序中输入用户名，并发送到服务器端，服务器端打印显示出连接信息，同时服务端发送给客户端验证问题：

```
wamdm@WAMDM52:~/taozi/ad_OS/socketnew$ ./server
server start...
taozi1 is requesting
send question
taozi2 is requesting
send question
```

```
wamdm@WAMDM51:~/taozinew/socketnew$ ./client 202.112.113.252
Please Input your name: taozi1
Please answer the question: 3+5=?
Input the answer:
```

```
wamdm@WAMDM51:~/taozinew/socketnew$ ./client 202.112.113.252
Please Input your name: taozi2
Please answer the question: 3+5=?
Input the answer:
```

4. taozi1先回答问题，但是答案错误：

```
wamdm@WAMDM51:~/taozinew/socketnew$ ./client 202.112.113.252
Please Input your name: taozi1
Please answer the question: 3+5=?
Input the answer: 7
Answer is wrong!,continue
Input the answer:
```

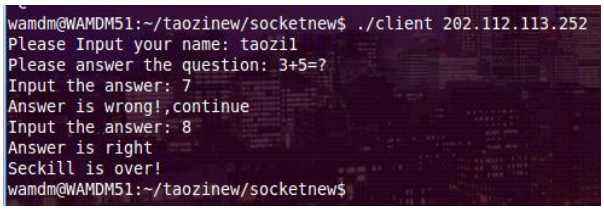
```
wamdm@WAMDM52:~/taozi/ad_OS/socketnew$ ./server
server start...
taozi1 is requesting
send question
taozi2 is requesting
send question
Answer 7 is wrong
```

4. taozi2回答问题，答案正确：

```
wamdm@WAMDM51:~/taozinew/socketnew$ ./client 202.112.113.252
Please Input your name: taozi2
Please answer the question: 3+5=?
Input the answer: 8
Answer is right
Congratulation,Seckill is successful!
wamdm@WAMDM51:~/taozinew/socketnew$
```

```
wamdm@WAMDM52:~/taozi/ad_OS/socketnew$ ./server
server start...
taozi1 is requesting
send question
taozi2 is requesting
send question
Answer 7 is wrong
Answer 8 is right
```

5. taozi1又回答问题，答案正确，但显示秒杀结束：



七、实验总结

①本实验主要是在理解C/S网络编程的基础上，基于TCP协议实现模拟秒杀的功能，利用socket实现客户与服务器之间的交互。

②实验可改进之处：a)秒杀是一个很讲究时间效率的活动，可能用UDP协议来实现省去建立连接的过程会更好；b)本实验中的验证问题只用了一道固定的题目来模拟，为了加大难度，这部分可以用一个记录在数据库或文件中的题库来实现。c)本实验采用2个客户端来模拟秒杀用户，所以服务器端的监听请求的队列长度设置得较小，当用户数量很多时，应适当增大监听队列长度，同时监听队列长度的设置要根据服务器本身的处理能力来协调。