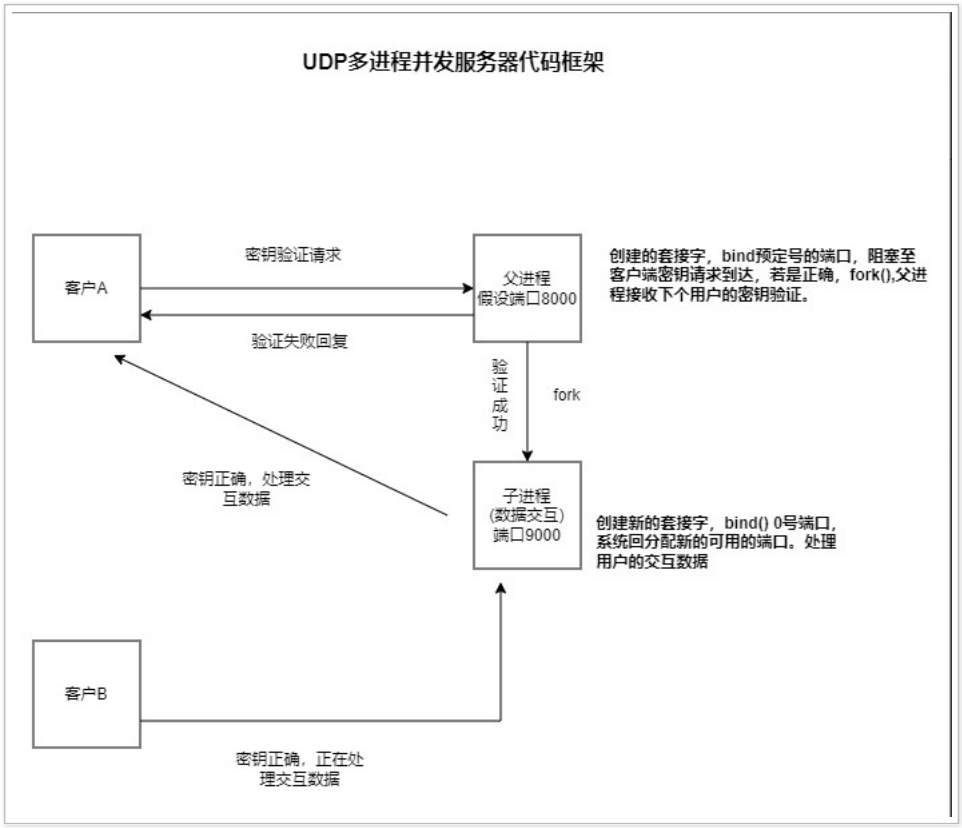
9.1 并发服务器之多进程并发_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 9.1 并发服务器之多进程并发涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

在网络程序里面, 通常都是一个服务器处理多个客户机。为了处理多个客户机的请求, 服务器端的程序有不同的处理方式。

- 常用的服务器模型
    - 迭代服务器
        - 大多数 UDP 都是迭代运行，服务器等用客户端的数据，收到数据后处理该数据，送回其应答，在等待下一个客户请求。
        - ```
          socket()
          bind()
          while(1)
          {
              recvfrom();
              process();
              sendto();
          }
          close();
          ```
    - 并发服务器
        - 并发服务器是指在同一个时刻可以响应多个客户端的请求。
        - 本质是创建多进程 / 多线程，对多数用户的信息进行处理。
        - UDP 协议一般默认是不支持多线程并发的，因为默认 UDP 服务器只有一个 sockfd，所有的客户端都是通过同一个 sockfd 进行通信的. udp 一个 socket，如何做到做并发呢?
        - ```
          sockfd = socket();
          bind();
          while(1)
          {
              recvfrom();
              ...
          }
          close();
          ```
        - 当 UDP 协议针对客户请求的处理需要消耗过长的时间时，我们期望 UDP 服务器具有某种形式的并发性。
- 场景设计
    - 多个 udp 客户端登录用户需要先验证密钥是否正确后，才能允许用户进行数据交互。假设密钥为 "root".
    - 服务器接收客户端信息的时候，需要考虑两种情况
    - - A 用户的密钥验证请求消息。
      - B 用户的数据交互接收消息。
      - 多个用户之间实现并发
    - 思维框架图



当 UDP 服务器与客户交互多个数据报。问题在于每个客户都是往服务器端的同一个的端口发送数据，并用的同一个 sockfd。并发服务器的每一个子进程如何正确区分每一个客户的数据报（涉及到进程的调度问题，如何避免一个子进程读取到不该它服务的客户发送来的数据报）。

解决的方法是服务器（知名端口）等待一下客户的到来，当一个客户到来后，记下其 IP 和 port，然后同理，服务器 fork 一个子进程，建立一个 socket 再 bind 一个随机端口，然后建立与客户的连接，并处理该客户的请求。父进程继续循环，等待下一个客户的到来。在 tftpd 中就是使用这种技术的。

```
sockfd = socket();
bind();

while(1)
{
    recvfrom();
    process();

    if(密钥验证正确)
    {
        if(fork() == 0)
        {
             close(sockfd);
             send();
             break;
        }
    }
    send();
}
```

udp_fork_server.c

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

#define LOGIN_SUCCESS 1
#define LOGIN_FAILURE 0
```

```
void printf_client_info(struct sockaddr_in *addr,char *buf)
{
        printf("==============================\n");
        printf("user IP : %s\n",inet_ntoa(addr->sin_addr));
        printf("user port : %d\n", ntohs(addr->sin_port));
        printf("user data : %s\n",buf);
}



int init_socket(const char *ip,const char *port)
{
        int sockfd = 0;
        struct sockaddr_in my_addr;
        socklen_t len =  sizeof(my_addr);


        sockfd = socket(AF_INET,SOCK_DGRAM ,0);
        if(sockfd < 0)
        {
                perror("Fail to socket!");
                exit(EXIT_FAILURE);
        }


        memset(&my_addr,0,sizeof(my_addr));
        my_addr.sin_family = AF_INET;
        my_addr.sin_port = htons(atoi(port));
        my_addr.sin_addr.s_addr = inet_addr(ip);

        if(bind(sockfd,(struct sockaddr *)&my_addr,len) < 0)
        {
                perror("Fail to bind");
                return -1;
        }
        return sockfd;
}

int user_login(const char *ip,const char *port)
{
    int n = 0;
    char buf[20] = {0};
    struct sockaddr_in client_addr;
    socklen_t len =  sizeof(client_addr);
    unsigned char login_flag;
    int sockfd;
    int new_sockfd;


    sockfd = init_socket(ip,port);


    while(1)
    {
        memset(buf,0,sizeof(buf));
        n = recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr *)&client_addr,&len);
        if(n < 0)
        {
                perror("Fail to sendto");
                exit(EXIT_FAILURE);
        }
        printf("key = %s\n",buf);

        login_flag = (strncmp(buf,"root",4) == 0) ? LOGIN_SUCCESS : LOGIN_FAILURE;


        if(login_flag == LOGIN_SUCCESS)
        {

            if(fork() == 0)
            {
                close(sockfd);


                new_sockfd = init_socket(ip,"0");
                sendto(new_sockfd,&login_flag,sizeof(login_flag),0,(struct sockaddr *)&client_addr
                break;
            }
        }else{

            sendto(sockfd,&login_flag,sizeof(login_flag),0,(struct sockaddr *)&client_addr,len);
        }

    }

    return new_sockfd;
```

```
}

void recv_data(int new_sockfd)
{
    int n = 0;
    char buf[1024] = {0};

    struct sockaddr_in client_addr;
    socklen_t len =  sizeof(client_addr);


    while(1)
    {
        memset(buf,0,sizeof(buf));
        n = recvfrom(new_sockfd,buf,sizeof(buf),0,(struct sockaddr *)&client_addr,&len);
        if(n < 0)
        {
            perror("Fail to sendto");
            exit(EXIT_FAILURE);
        }

        printf_client_info(&client_addr,buf);

        if(strncmp(buf,"quit",4) == 0)
                break;
    }

    close(new_sockfd);
    exit(EXIT_SUCCESS);
    return ;
}

void sig_handler(int signum)
{
    waitpid(-1,NULL,WNOHANG);
    printf("recv singnum = %d zombie\n",signum);
    return ;
}


int main(int argc, const char *argv[])
{
    int sockfd;
    unsigned char login_flag;
    if(argc < 3)
    {
        fprintf(stderr,"Usage : %s ip port!\n",argv[0]);
        exit(EXIT_FAILURE);
    }


    if(signal(SIGCHLD,sig_handler) == SIG_ERR)
    {
        perror("Fail to single\n");
        return -1;
    }


    sockfd = user_login(argv[1],argv[2]);


    recv_data(sockfd);

    return 0;
}
```

udp_client.c

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

#define LOGIN_SUCCESS 1
#define LOGIN_FAILURE 0

void user_login(int sockfd,struct sockaddr_in *addr,struct sockaddr_in *new_addr,int len)
{
    int n = 0;
```

```
    char buf[1024] = {0};
    unsigned char flag = LOGIN_FAILURE;


    while(1)
    {
        putchar('>');
        memset(buf,0,sizeof(buf));
        fgets(buf,sizeof(buf),stdin);
        buf[strlen(buf) - 1] = '\0';

        n = sendto(sockfd,buf,strlen(buf),0,(struct sockaddr *)addr,len);
        if(n < 0)
        {
            perror("Fail to sendto");
            exit(EXIT_FAILURE);
        }

        recvfrom(sockfd,&flag,sizeof(flag),0,(struct sockaddr *)new_addr,&len);



        if(flag == LOGIN_SUCCESS)
                break;

    }
    return ;
}

void send_message(int sockfd,struct sockaddr_in *addr,int addr_len)
{
    int n = 0;
    char buf[1024] = {0};


    while(1)
    {
        printf("Input : ");
        memset(buf,0,sizeof(buf));
        fgets(buf,sizeof(buf),stdin);
        buf[strlen(buf) - 1] = '\0';

        n = sendto(sockfd,buf,strlen(buf),0,(struct sockaddr *)addr,addr_len);
        if(n < 0)
        {
            perror("Fail to sendto");
            exit(EXIT_FAILURE);
        }
        if(strncmp(buf,"quit",4) == 0)
                break;
    }
    return ;
}


int main(int argc, const char *argv[])
{
    int sockfd = 0;
    struct sockaddr_in peer_addr;
    struct sockaddr_in server_addr;
    socklen_t len =  sizeof(peer_addr);

    if(argc < 3)
    {
        fprintf(stderr,"Usage : %s ip port!\n",argv[0]);
        exit(EXIT_FAILURE);
    }


    sockfd = socket(AF_INET,SOCK_DGRAM ,0);
    if(sockfd < 0)
    {
        perror("Fail to socket!");
        exit(EXIT_FAILURE);
    }


    memset(&peer_addr,0,sizeof(peer_addr));
    peer_addr.sin_family = AF_INET;
    peer_addr.sin_port = htons(atoi(argv[2]));
    peer_addr.sin_addr.s_addr = inet_addr(argv[1]);


    memset(&server_addr,0,sizeof(server_addr));
    user_login(sockfd,&peer_addr,&server_addr,len);
```

```
    send_message(sockfd,&server_addr,len);


    close(sockfd);

    return 0;
}
```

练习：

大家自己看懂老师的代码后，自己编写 udp 多进程并发服务器的代码。并和网络调试助手调试成功后。

把服务器相关代码上传即可。

全文完