

TCP/IP网络编程(十二)

笔记本： 网络编程

创建时间： 2018/11/13 10:03

更新时间： 2018/11/13 11:01

作者： xiangkang94@outlook.com

标签： 第十二章(I/O复用)

1. 多进程服务器的缺点

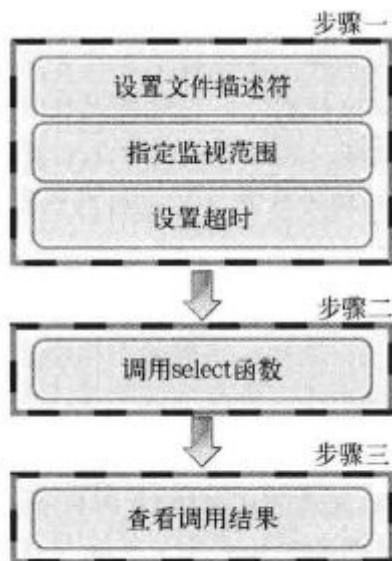
- 只要有客户端连接就会创建新的进程，需要大量的运算和内存空间作为代价
- 数据交互要求采用较为复杂的方法(IPC)

2. 理解复用

- 纸杯电话的例子
- 一定程度的时分复用(不是所有时刻都同时说话)
- 一定程度的频分复用(每个人的声音频率不一样)

3. 理解select函数

- 将多个文件描述符集中在一起统一监视
- 监视项目有：
 - 是否存在套接字接收数据；
 - 无需阻塞传输数据的套接字有哪些；
 - 哪些套接字发生异常；



■ 图12-5 select函数调用过程

○ 设置文件描述符

- FD_ZERO(fd_set *fdset) //将fd_set变量的所有位初始化0
- FD_SET(int fd, fd_set *fdset) //注册文件描述符fd到fdset (以位数组的形式注册)
- FD_CLR(int fd, fd_set *fdset) //清除文件描述符fd
- FD_ISSET(int fd, fd_set *fdset) //判断是否包含

○ 设置检查范围和超时

```
#include <sys/select.h>
#include <sys/time.h>
```

```
int select(int maxfd, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval * timeout)
```

成功返回大于0的值，超时返回0，失败返回-1

maxfd: 监视对象文件描述符数量 (最大文件描述符+1)

readset: 是否存在待读取数据的文件描述符

writeset: 是否可传输无阻塞数据的文件描述符

exceptset: 是否发生异常的文件描述符

timeout: 超时信息 (如果select发生了阻塞，那么就通过设置timeout防止这种情况)

监视范围：与第一个参数有关，最大的文件描述符值加1（加1,是因为文件描述符从0开始）。

超时：

```
struct timeval{
    long tv_sec; //sec
    long tv_usec; // microsec
}
```

4. 实现I/O复用回声服务器端

```
#include <...h>
#define BUF_SIZE 100
void error_handling(char *buf);

int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_addr, clnt_addr;
    struct timeval timeout;
    fd_set reads, cpy_reads;

    socklen_t adr_sz;
    int fd_max, str_len, fd_num, i;
    char buf[BUF_SIZE];
    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))== -1)
        error_handling("bind() error");
    if(listen(serv_sock, 5)== -1)
        error_handling("listen() error");

    FD_ZERO(&reads);
    FD_SET(serv_sock, &reads);
    fd_max=serv_sock;

    while(1)
    {
        cpy_reads=reads;    //由于调用select之后，除发生了变化的描述符外，剩余的所有位都将置0,为了记住初始值，需要
复制
        timeout.tv_sec=5;    //在循环内部设置timeout的原因在于select执行完之后，timeout的值会变成超时前剩余时间
        timeout.tv_usec=5000;

        if((fd_num=select(fd_max+1, &cpy_reads, 0, 0, &timeout))== -1) //侦听cpy_reads注册的文件描述符中是否存在
在待读取
            break;

        if(fd_num==0)
            continue;

        for(i=0; i<fd_max+1; i++)
        {
            if(FD_ISSET(i, &cpy_reads))    //查找发生变化的(有接收数据的套接字)文件描述符
            {
                if(i==serv_sock)    // connection request!
                {
                    adr_sz=sizeof(clnt_addr);
                    clnt_sock=
                        accept(serv_sock, (struct sockaddr*)&clnt_addr, &adr_sz);
                    FD_SET(clnt_sock, &reads);    //连接成功后，需要注册与客户端连接套字到reads中去，用来侦听后续数
据的到来
                    if(fd_max<clnt_sock)
```

```

        fd_max=clnt_sock;
        printf("connected client: %d \n", clnt_sock);
    }
    else    // read message!
    {
        str_len=read(i, buf, BUF_SIZE);
        if(str_len==0)    // close request!
        {
            FD_CLR(i, &reads);
            close(i);
            printf("closed client: %d \n", i);
        }
        else
        {
            write(i, buf, str_len);    // echo!
        }
    }
}
}
close(serv_sock);
return 0;
}

void error_handling(char *buf)
{
    fputs(buf, stderr);
    fputc('\n', stderr);
    exit(1);
}

```