# 网络编程第5天

## 内容回顾

poll

输入和输出分离, 输入用events, 输出用revents

epoll

ET模式是写一次通知一次, 若写的数据多, 读的数据少, 则不会再通知, 直到下一次有写事件发生.

思考题? epoll监控监听文件描述符可以设置成ET模式吗??

答案: 可以. 但是如果设置成ET模式以后, 当调用epoll\_wait函数的时候, 每次只能accept一个连接(该连接在已连接队列当中, 而调用一次accept只能已连接队列中获取一个连接), 如果同时有多个连接到来, 就得epoll\_wait再次返回之后才能继续accept下一个连接, 所以如果设置成了ET模式且有多个连接请求的话, 应该将accept写在循环当中, 一次epoll\_wait后循环accept所有的连接.

所以一般不会在epoll中将监听的文件描述符设置为ET模式, 使用默认的LT模式即可; 而对通信的文件描述符一般采用非阻塞模式的ET模式.

## 学习目标

* 了解线程池模型的设计思想
* 能看懂线程池实现源码
* 掌握tcp和udp的优缺点和使用场景
* 说出udp服务器通信流程
* 说出udp客户端通信流程
* 独立实现udp服务器代码
* 独立实现udp客户端代码
* 熟练掌握本地套接字进行本地进程通信

## 线程池

什么是线程池?

是一个抽象的概念, 若干个线程组合到一起, 形成线程池.

为什么需要线程池?

多线程版服务器一个客户端就需要创建一个线程! 若客户端太多, 显然不太合适.

什么时候需要创建线程池呢？简单的说，如果一个应用需要频繁的创建和销毁线程，而任务执行的时间又非常短，这样线程创建和销毁的带来的开销就不容忽视，这时也是线程池该出场的机会了。如果线程创建和销毁时间相比任务执行时间可以忽略不计，则没有必要使用线程池了。

实现的时候类似于生产者和消费者.

线程池和任务池:

任务池相当于共享资源, 所以需要使用互斥锁, 当任务池中没有任务的时候需要让线程阻塞, 所以需要使用条件变量.

如何让线程执行不同的任务?

使用回到函数, 在任务中设置任务执行函数, 这样可以起到不同的任务执行不同的函数.

通过阅读线程池代码思考如下问题?

* 熟悉结构体 threadpool\_t
* 线程池如何创建起来? 各种初始化，malloc，pthread\_create，pthread\_cond\_init pthread\_mutex\_init
* 线程池内都有几类线程? 2类：管理线程+工作线程
* 管理者线程的任务是什么?任务如何实现? 任务是添加线程或者删除线程，通过2个算法，删除线程 wait\_exit\_thr\_num = 10
* 工作线程如何工作? 等待有任务，抢到任务，修改busy\_thr\_num ++ 执行任务 修改 busy\_thr\_num --
* 线程池是如何销毁的? 自爆shutdown 诱杀！

讲解代码threadpoolsimple.c

讲解代码 pthreadpool.c

## UDP通信

TCP：传输控制协议, 面向连接的，稳定的，可靠的，安全的数据流传递

稳定和可靠: 丢包重传

数据有序: 序号和确认序号

流量控制: 滑动窗口

UDP：用户数据报协议

面向无连接的，不稳定，不可靠，不安全的数据报传递---更像是收发短信

UDP传输不需要建立连接，传输效率更高，在稳定的局域网内环境相对可靠

UDP通信相关函数介绍:

ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

函数说明: 接收消息

参数说明:

* sockfd 套接字
* buf 要接受的缓冲区
* len 缓冲区的长度
* flags 标志位 一般填0
* src\_addr 原地址 传出参数
* addrlen 发送方地址长度
* 返回值

成功: 返回读到的字节数

失败: 返回 -1 设置errno

调用该函数相当于TCP通信的recv+accept函数

ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

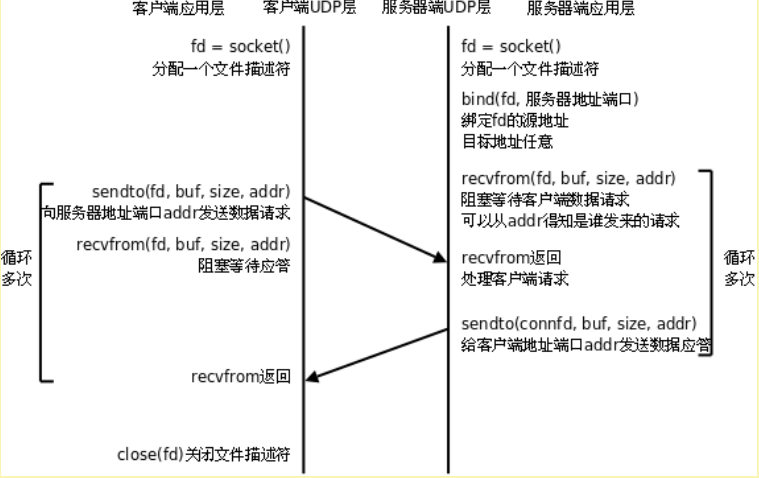
函数说明: 发送数据

参数说明:

* sockfd 套接字
* dest\_addr 目的地址
* addrlen 目的地址长度
* 返回值

成功: 返回写入的字节数

失败: 返回-1，设置errno



UDP的服务器编码流程：

* 创建套接字 type=SOCK\_DGRAM
* 绑定ip和端口
* while(1)

{

收发消--recvfrom

发消息--sendto

}

* 关闭套接字--close

UDP客户端流程：

* 创建套接字--socket
* while(1)

{

收发消--recvfrom

发消息--sendto

}

* 关闭套接字--close

编写udp代码并进行测试

测试:

多开器几个客户端经过测试表明:, udp天然支持多客户端, 这点和TCP不同, TCP需要维护连接.

使用nc命令进行测试: nc -u 127.1 8888

## 本地socket通信

回顾一些linux系统有哪些文件类型?

回顾一些linux系统下有哪些常见的IPC机制?

通过查询: man 7 unix 可以查到unix本地域socket通信相关信息:

#include <sys/socket.h>

#include <sys/un.h>

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

函数说明: 创建本地域socket

函数参数:

domain: AF\_UNIX or AF\_LOCAL

type: SOCK\_STREAM或者SOCK\_DGRAM

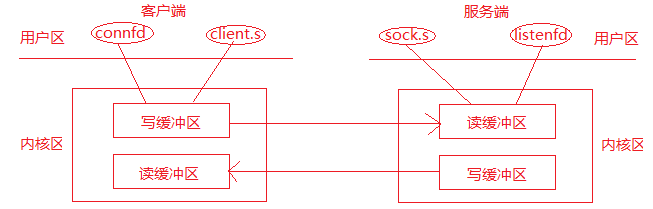
protocol: 0 表示使用默认协议

函数返回值:

成功: 返回文件描述符.

失败: 返回-1, 并设置errno值.

创建socket成功以后, 会在内核创建缓冲区, 下图是客户端和服务端内核缓冲区示意图.



int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

函数说明: 绑定套接字

函数参数:

socket: 由socket函数返回的文件描述符

addr: 本地地址

addlen: 本地地址长度

函数返回值:

成功: 返回文件描述符.

失败: 返回-1, 并设置errno值.

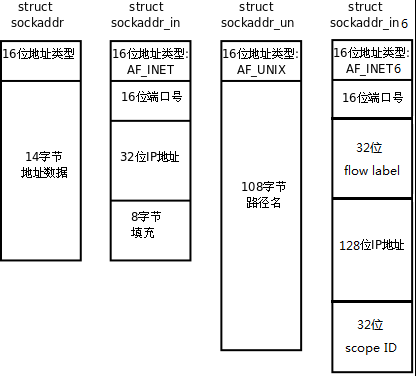
需要注意的是: bind函数会自动创建socket文件, 若在调用bind函数之前socket文件已经存在, 则调用bind会报错, 可以使用unlink函数在bind之前先删除文件.

struct sockaddr\_un {

sa\_family\_t sun\_family; /\* AF\_UNIX or AF\_LOCAL\*/

char sun\_path[108]; /\* pathname \*/

};



通过man 2 bind, 可以查看bind函数的相关信息, 后面还有示例代码, 可以参考.

本地套接字服务器的流程：

* 可以使用TCP的方式, 必须按照tcp的流程
* 也可以使用UDP的方式, 必须按照udp的流程

tcp的本地套接字服务器流程：

* 创建套接字 socket(AF\_UNIX,SOCK\_STREAM,0)
* 绑定 struct sockaddr\_un &强转
* 侦听 listen
* 获得新连接 accept
* 循环通信 read-write
* 关闭文件描述符 close

tcp本地套接字客户端流程：

* 调用socket创建套接字
* 调用bind函数将socket文件描述和socket文件进行绑定.

不是必须的, 若无显示绑定会进行隐式绑定，但服务器不知道谁连接了.

* 调用connect函数连接服务端
* 循环通信read-write
* 关闭文件描述符 close

编写代码并进行测试

测试客户端工具:

man nc

-U Specifies to use UNIX-domain sockets.

例如: nc -U sock.s

size = offsetof(struct sockaddr\_un, sun\_path) +strlen(un.sun\_path);

#define offsetof(type, member) ((int)&((type \*)0)->member)