# 网络编程第4天

# 学习目标

1 了解poll函数

2 熟练使用epoll多路IO模型

3 了解epoll ET/LT触发模式并实现

4 理解epoll边缘非阻塞模式并实现

5 了解epoll反应堆模型设计思想

6 能看懂epoll反应堆模型的实现代码

## 多路IO-poll

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);

函数说明: 跟select类似, 监控多路IO, 但poll不能跨平台.

参数说明:

fds: 传入传出参数, 实际上是一个结构体数组

fds.fd: 要监控的文件描述符

fds.events:

POLLIN---->读事件

POLLOUT---->写事件

fds.revents: 返回的事件

nfds: 数组实际有效内容的个数

timeout: 超时时间, 单位是毫秒.

-1:永久阻塞, 直到监控的事件发生

0: 不管是否有事件发生, 立刻返回

>0: 直到监控的事件发生或者超时

返回值:

成功:返回就绪事件的个数

失败: 返回-1

若timeout=0, poll函数不阻塞,且没有事件发生, 此时返回-1, 并且errno=EAGAIN, 这种情况不应视为错误.

struct pollfd

{

int fd; /\* file descriptor \*/ 监控的文件描述符

short events; /\* requested events \*/ 要监控的事件---不会被修改

short revents; /\* returned events \*/ 返回发生变化的事件 ---由内核返回

};

说明:

1 当poll函数返回的时候, 结构体当中的fd和events没有发生变化, 究竟有没有事件发生由revents来判断, 所以poll是请求和返回分离.

2 struct pollfd结构体中的fd成员若赋值为-1, 则poll不会监控.

3 相对于select, poll没有本质上的改变; 但是poll可以突破1024的限制.

在/proc/sys/fs/file-max查看一个进程可以打开的socket描述符上限.

如果需要可以修改配置文件: /etc/security/limits.conf

加入如下配置信息, 然后重启终端即可生效.

\* soft nofile 1024

\* hard nofile 100000

soft和hard分别表示ulimit命令可以修改的最小限制和最大限制

## 多路IO-epoll

将检测文件描述符的变化委托给内核去处理, 然后内核将发生变化的文件描述符对应的事件返回给应用程序.

函数介绍:

int epoll\_create(int size);

函数说明: 创建一个树根

参数说明:

size: 最大节点数, 此参数在linux 2.6.8已被忽略, 但必须传递一个大于0的数.

返回值:

成功: 返回一个大于0的文件描述符, 代表整个树的树根.

失败: 返回-1, 并设置errno值.

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

函数说明: 将要监听的节点在epoll树上添加, 删除和修改

参数说明:

epfd: epoll树根

op:

EPOLL\_CTL\_ADD: 添加事件节点到树上

EPOLL\_CTL\_DEL: 从树上删除事件节点

EPOLL\_CTL\_MOD: 修改树上对应的事件节点

fd: 事件节点对应的文件描述符

event: 要操作的事件节点

typedef union epoll\_data {

void \*ptr;

int fd;

uint32\_t u32;

uint64\_t u64;

} epoll\_data\_t;

struct epoll\_event {

uint32\_t events; /\* Epoll events \*/

epoll\_data\_t data; /\* User data variable \*/

};

event.events常用的有:

EPOLLIN: 读事件

EPOLLOUT: 写事件

EPOLLERR: 错误事件

EPOLLET: 边缘触发模式

event.fd: 要监控的事件对应的文件描述符

int epoll\_wait(int epfd, struct epoll\_event \*events, int maxevents, int timeout);

函数说明:等待内核返回事件发生

参数说明:

epfd: epoll树根

events: 传出参数, 其实是一个事件结构体数组

maxevents: 数组大小

timeout:

-1: 表示永久阻塞

0: 立即返回

>0: 表示超时等待事件

返回值:

成功: 返回发生事件的个数

失败: 若timeout=0, 没有事件发生则返回; 返回-1, 设置errno值,

epoll\_wait的events是一个传出参数, 调用epoll\_ctl传递给内核什么值, 当epoll\_wait返回的时候, 内核就传回什么值,不会对struct event的结构体变量的值做任何修改.

代码思路

编写代码测试

相关总结

## 进阶epoll

### 介绍epoll的两种工作模式

epoll的两种模式ET和LT模式

水平触发: 高电平代表1

只要缓冲区中有数据, 就一直通知

边缘触发: 电平有变化就代表1

缓冲区中有数据只会通知一次, 之后再有数据才会通知.(若是读数据的时候没有读完, 则剩余的数据不会再通知, 直到有新的数据到来)

边缘非阻塞模式: 提高效率

### 用实验验证LT和ET模式

ET模式由于只通知一次, 所以在读的时候要循环读, 直到读完, 但是当读完之后read就会阻塞, 所以应该将该文件描述符设置为非阻塞模式(fcntl函数).

read函数在非阻塞模式下读的时候, 若返回-1, 且errno为EAGAIN, 则表示当前资源不可用, 也就是说缓冲区无数据(缓冲区的数据已经读完了); 或者当read返回的读到的数据长度小于请求的数据长度时，就可以确定此时缓冲区中已没有数据可读了，也就可以认为此时读事件已处理完成。

## epoll反应堆

反应堆: 一个小事件触发一系列反应.

epoll反应堆的思想: c++的封装思想(把数据和操作封装到一起)

--将描述符,事件,对应的处理方法封装在一起

--当描述符对应的事件发生了, 自动调用处理方法(其实原理就是回调函数)

**typedef union epoll\_data {**

**void \*ptr;**

**int fd;**

**uint32\_t u32;**

**uint64\_t u64;**

**} epoll\_data\_t;**

**struct epoll\_event {**

**uint32\_t events; /\* Epoll events \*/**

**epoll\_data\_t data; /\* User data variable \*/**

**};**

epoll反应堆的核心思想是: 在调用epoll\_ctl函数的时候, 将events上树的时候,利用**epoll\_data\_t的ptr成员, 将一个文件描述符,事件和回调函数封装成一个结构体, 然后让ptr指向这个结构体, 然后调用epoll\_wait函数返回的时候, 可以得到具体的events, 然后获得events结构体中的events.data.ptr指针, ptr指针指向的结构体中有回调函数, 最终可以调用这个回调函数.**

讲解代码-讲思路

## 作业

预习: 线程池, UDP, 本地套接字

思考: 能够将监听文件描述符改成ET模式