ljx0305的专栏

个人资料

ljx0305

+ 加关注 发私信

访问： 664086次
积分： 7233
等级： BLOG 6
排名： 第1776名
原创： 80篇 转载： 412篇
译文： 1篇 评论： 85条

文章搜索

文章分类

ACE (1)
C (81)
C++ (170)
COM组件 (1)
DLL (8)
excel使用 (1)
Linux (43)
linux下软件安装 (31)
Linux程序及使用 (106)
MFC (37)
MySql (6)
Oracle体系结构 (20)
Sqlite DB (1)
TCP协议 (1)
UDP协议 (3)
VC (79)
XML (4)
其它 (14)
其它杂谈 (1)
数据结构 (5)
数码相机 (2)

转 epoll使用详解（精髓）

标签： events socket struct 网络 服务器 linux

2009-04-11 16:34 125990人阅读 评论(20) 收藏 举报

分类： Linux程序及使用（105）

epoll - I/O event notification facility

在linux的网络编程中，很长的时间都在使用select来做事件触发。在linux新的内核中，有了一种替换它的机制，就是epoll。
相比于select，epoll最大的好处在于它不会随着监听fd数目的增长而降低效率。因为在内核中的select实现中，它是采用轮询来处理的，轮询的fd数目越多，自然耗时越多。并且，在linux/posix_types.h头文件有这样的声明：

#define __FD_SETSIZE 1024

表示select最多同时监听1024个fd，当然，可以通过修改头文件再重编译内核来扩大这个数目，但这似乎并不治本。

epoll的接口非常简单，一共就三个函数：

1. int epoll_create(int size);

创建一个epoll的句柄，size用来告诉内核这个监听的数目一共有多大。这个参数不同于select()中的第一个参数，给出最大监听的fd+1的值。需要注意的是，当创建好epoll句柄后，它就是会占用一个fd值，在linux下如果查看/proc/进程id/fd/，是能够看到这个fd的，所以在使用完epoll后，必须调用close()关闭，否则可能导致fd被耗尽。

2. int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

epoll的事件注册函数，它不同与select()是在监听事件时告诉内核要监听什么类型的事件，而是在这里先注册要监听的事件类型。第一个参数是epoll_create()的返回值，第二个参数表示动作，用三个宏来表示：

EPOLL_CTL_ADD：注册新的fd到epfd中；

EPOLL_CTL_MOD：修改已经注册的fd的监听事件；

EPOLL_CTL_DEL：从epfd中删除一个fd；

第三个参数是需要监听的fd，第四个参数是告诉内核需要监听什么事，struct epoll_event结构如下：

```
typedef union epoll_data {
    void *ptr;
    int fd;
    __uint32_t u32;
    __uint64_t u64;
} epoll_data_t;
```

汇编 (1)
汉字的编码与字模点阵 (1)
源代码地址 (1)
算法 (2)
算法 (3)
编译，链接，调试 (0)
网站链接 (0)
网络编程 (18)
计算几何 (1)
问题总结 (5)
Qt (3)
嵌入式 (1)
GoAhead (1)

文章存档

2014年04月 (3)
2014年02月 (1)
2013年12月 (1)
2013年07月 (5)
2013年06月 (3)

阅读排行

epoll使用详解（精髓） (125878)
RedHat Linux安装Oracle (10424)
sigsetjmp,siglongjmp的使 (8613)
滑动窗口协议 (8274)
清除系统垃圾的bat文件 (8184)
如何快速算出一个日期是 (6464)
深入解析数码相机CCD场 (6438)
RedHat Linux 9.0 安装教 (6331)
smartdrv.exe的使用及简 (5891)
linux下C程序获取绝对路 (5340)

评论排行

epoll使用详解（精髓） (20)
Big&Little (6)
深入探讨MFC消息循环和 (6)
信号基本原理 (4)
Sqlite DB使用例子 (3)
线程池原理及创建（C++ (3)
Windows CE 进程、线程 (2)
MD5之C语言源代码 及 使 (2)
汉字拼音及拼音码获取 (2)
R6034 又来了. (2)

推荐文章

*Android官方开发文档Training系列课程中文版：高效显示位图之在非UI线程中处理图片
*Binder工作机制
* Java Web基础知识之Filter：过滤一切你不想看到的事情/a>
*Untiy Native Render Plugin在VR中的绘制(二): 透明排序
*随机过程--Metropolis-Hastings算法
*Fresco图片库研读分析

```
struct epoll_event {
    __uint32_t events; /* Epoll events */
    epoll_data_t data; /* User data variable */
};
```

events可以是以下几个宏的集合：
EPOLLIN ： 表示对应的文件描述符可以读（包括对端SOCKET正常关闭）；
EPOLLOUT：表示对应的文件描述符可以写；
EPOLLPRI：表示对应的文件描述符有紧急的数据可读（这里应该表示有带外数据到来）；
EPOLLERR：表示对应的文件描述符发生错误；
EPOLLHUP：表示对应的文件描述符被挂断；
EPOLLET： 将EPOLL设为边缘触发(Edge Triggered)模式，这是相对于水平触发(Level Triggered)来说的。
EPOLLONESHOT：只监听一次事件，当监听完这次事件之后，如果还需要继续监听这个socket的话，需要再次把这个socket加入到EPOLL队列里

3. int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);
等待事件的产生，类似于select()调用。参数events用来从内核得到事件的集合，maxevents告之内核这个events有多大，这个 maxevents的值不能大于创建epoll_create()时的size，参数timeout是超时时间（毫秒，0会立即返回，-1将不确定，也有说法说是永久阻塞）。该函数返回需要处理的事件数目，如返回0表示已超时。

4、关于ET、LT两种工作模式：
可以得出这样的结论:
ET模式仅当状态发生变化的时候才获得通知,这里所谓的状态的变化并不包括缓冲区中还有未处理的数据,也就是说,如果要采用ET模式,需要一直read/write直到出错为止,很多人反映为什么采用ET模式只接收了一部分数据就再也得不到通知了,大多因为这样;而LT模式是只要有数据没有处理就会一直通知下去的.

那么究竟如何来使用epoll呢？其实非常简单。
通过在包含一个头文件#include <sys/epoll.h> 以及几个简单的API将可以大大的提高你的网络服务器的支持人数。

首先通过create_epoll(int maxfds)来创建一个epoll的句柄，其中maxfds为你epoll所支持的最大句柄数。这个函数会返回一个新的epoll句柄，之后的所有操作将通过这个句柄来进行操作。在用完之后，记得用close()来关闭这个创建出来的epoll句柄。

之后在你的网络主循环里面，每一帧的调用epoll_wait(int epfd, epoll_event events, int max events, int timeout)来查询所有的网络接口，看哪一个可以读，哪一个可以写了。基本的语法为：
nfds = epoll_wait(kdpfd, events, maxevents, -1);
其中kdpfd为用epoll_create创建之后的句柄，events是一个epoll_event*的指针，当epoll_wait这个函数操作成功之后，epoll_events里面将储存所有的读写事件。max_events是当前需要监听的所有socket句柄数。最后一个timeout是 epoll_wait的超时，为0的时候表示马上返回，为-1的时候表示一直等下去，直到有事件范围，为任意正整数的时候表示等这么长的时间，如果一直没有事件，则范围。一般如果网络主循环是单独的线程的话，可以用-1来等，这样可以保证一些效率，如果是和主逻辑在同一个线程的话，则可以用0来保证主循环的效率。

epoll_wait范围之后应该是一个循环，遍利所有的事件。

几乎所有的epoll程序都使用下面的框架：

```
for( ; ; )
{
    nfds = epoll_wait(epfd,events,20,500);
    for(i=0;i<nfds;++i)
    {
        if(events[i].data.fd==listenfd) //有新的连接
        {
            connfd = accept(listenfd,(sockaddr *)&clientaddr, &clilen); //accept这个连接
            ev.data.fd=connfd;
            ev.events=EPOLLIN|EPOLLET;
            epoll_ctl(epfd,EPOLL_CTL_ADD,connfd,&ev); //将新的fd添加到epoll的监听队列中
        }
        else if( events[i].events&EPOLLIN ) //接收到数据，读socket
        {
            n = read(sockfd, line, MAXLINE)) < 0    //读
            ev.data.ptr = md;     //md为自定义类型，添加数据
            ev.events=EPOLLOUT|EPOLLET;
            epoll_ctl(epfd,EPOLL_CTL_MOD,sockfd,&ev);//修改标识符，等待下一个循环时发送数据，异步处理的精髓
        }
        else if(events[i].events&EPOLLOUT) //有数据待发送，写socket
        {
            struct myepoll_data* md = (myepoll_data*)events[i].data.ptr;    //取数据
            sockfd = md->fd;
            send( sockfd, md->ptr, strlen((char*)md->ptr), 0 );        //发送数据
            ev.data.fd=sockfd;
            ev.events=EPOLLIN|EPOLLET;
            epoll_ctl(epfd,EPOLL_CTL_MOD,sockfd,&ev); //修改标识符，等待下一个循环时接收数据
        }
        else
        {
            //其他的处理
        }
    }
}
```

下面给出一个完整的服务器端例子：

```
#include <iostream>
#include <sys/socket.h>
#include <sys/epoll.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

using namespace std;

#define MAXLINE 5
```

维C世界
中国红客基地

Linux

linux 线程 进程经典文章
进程的状态转换
编译OCI程序出现的问题
LinuxSir网站
嵌入式开发网 资料下载网
GNU-make v3.80中文版

P2P

P2P技术(P2P行业第一中文门户网)

VC

VC知识库在线杂志
问专家
fisher_jiang的专栏 有c++
让UI开发轻松而快乐，用SonicUI引擎实现常见UI效果 (RSS)

```
#define OPEN_MAX 100
#define LISTENQ 20
#define SERV_PORT 5000
#define INFTIM 1000

void setnonblocking(int sock)
{
    int opts;
    opts=fcntl(sock,F_GETFL);
    if(opts<0)
    {
        perror("fcntl(sock,GETFL)");
        exit(1);
    }
    opts = opts|O_NONBLOCK;
    if(fcntl(sock,F_SETFL,opts)<0)
    {
        perror("fcntl(sock,SETFL,opts)");
        exit(1);
    }
}

int main(int argc, char* argv[])
{
    int i, maxi, listenfd, connfd, sockfd,epfd,nfds, portnumber;
    ssize_t n;
    char line[MAXLINE];
    socklen_t clilen;


    if ( 2 == argc )
    {
        if( (portnumber = atoi(argv[1])) < 0 )
        {
            fprintf(stderr,"Usage:%s portnumber/a/n",argv[0]);
            return 1;
        }
    }
    else
    {
        fprintf(stderr,"Usage:%s portnumber/a/n",argv[0]);
        return 1;
    }



    //声明epoll_event结构体的变量,ev用于注册事件,数组用于回传要处理的事件

    struct epoll_event ev,events[20];
    //生成用于处理accept的epoll专用的文件描述符

    epfd=epoll_create(256);
    struct sockaddr_in clientaddr;
    struct sockaddr_in serveraddr;
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    //把socket设置为非阻塞方式

    //setnonblocking(listenfd);

    //设置与要处理的事件相关的文件描述符

    ev.data.fd=listenfd;
    //设置要处理的事件类型

    ev.events=EPOLLIN|EPOLLET;
```

```
    //ev.events=EPOLLIN;

    //注册epoll事件

    epoll_ctl(epfd,EPOLL_CTL_ADD,listenfd,&ev);
    bzero(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    char *local_addr="127.0.0.1";
    inet_aton(local_addr,&(serveraddr.sin_addr));//htons(portnumber);

    serveraddr.sin_port=htons(portnumber);
    bind(listenfd,(sockaddr *)&serveraddr, sizeof(serveraddr));
    listen(listenfd, LISTENQ);
    maxi = 0;
    for ( ; ; ) {
        //等待epoll事件的发生

        nfds=epoll_wait(epfd,events,20,500);
        //处理所发生的所有事件

        for(i=0;i<nfds;++i)
        {
            if(events[i].data.fd==listenfd)//如果新监测到一个SOCKET用户连接到了绑定的SOCKET端口，建立新的连接。

            {
                connfd = accept(listenfd,(sockaddr *)&clientaddr, &clilen);
                if(connfd<0){
                    perror("connfd<0");
                    exit(1);
                }
                //setnonblocking(connfd);

                char *str = inet_ntoa(clientaddr.sin_addr);
                cout << "accapt a connection from " << str << endl;
                //设置用于读操作的文件描述符

                ev.data.fd=connfd;
                //设置用于注测的读操作事件

                ev.events=EPOLLIN|EPOLLET;
                //ev.events=EPOLLIN;

                //注册ev

                epoll_ctl(epfd,EPOLL_CTL_ADD,connfd,&ev);
            }
            else if(events[i].events&EPOLLIN)//如果是已经连接的用户，并且收到数据，那么进行读入。

            {
                cout << "EPOLLIN" << endl;
                if ( (sockfd = events[i].data.fd) < 0)
                    continue;
                if ( (n = read(sockfd, line, MAXLINE)) < 0) {
                    if (errno == ECONNRESET) {
                        close(sockfd);
                        events[i].data.fd = -1;
                    } else
                        std::cout<<"readline error"<<std::endl;
                } else if (n == 0) {
                    close(sockfd);
                    events[i].data.fd = -1;
                }
                line[n] = '/0';
```

```
                cout << "read " << line << endl;
                //设置用于写操作的文件描述符

                ev.data.fd=sockfd;
                //设置用于注测的写操作事件

                ev.events=EPOLLOUT|EPOLLET;
                //修改sockfd上要处理的事件为EPOLLOUT

                //epoll_ctl(epfd,EPOLL_CTL_MOD,sockfd,&ev);

            }
            else if(events[i].events&EPOLLOUT) // 如果有数据发送

            {
                sockfd = events[i].data.fd;
                write(sockfd, line, n);
                //设置用于读操作的文件描述符

                ev.data.fd=sockfd;
                //设置用于注测的读操作事件

                ev.events=EPOLLIN|EPOLLET;
                //修改sockfd上要处理的事件为EPOLIN

                epoll_ctl(epfd,EPOLL_CTL_MOD,sockfd,&ev);
            }
        }
    }
    return 0;
}
```

客户端直接连接到这个服务器就好了。。

引用：http://blog.chinaunix.net/u/16292/showart_1844376.html

顶 4　踩 0

上一篇 C++中重载强制类型转换

下一篇 什么是Core Dump?

我的同类文章

Linux程序及使用（105）

- 不同平台字节序影响位字段… 2013-01-07 阅读 333
- errno与多线程 2012-12-28 阅读 1525
- POSIX线程专有数据的空间… 2012-12-28 阅读 495
- 守护进程的编写 2012-11-08 阅读 416
- Linux添加环境变量与GCC… 2012-11-07 阅读 398
- Posix线程编程指南(2) 2012-12-28 阅读 304
- errno多线程安全 2012-12-28 阅读 445
- fork两次如何避免僵尸进程 2012-11-08 阅读 426
- linux C 正则表达式 2012-11-08 阅读 764
- Nginx源码剖析之内存池，与… 2012-09-17 阅读 523

更多文章

猜你在找

- Android开发精品课程【Java核心知识】
- 韦东山嵌入式Linux第一期视频
- Linux高薪入门实战精品（上）
- Qt网络编程实战之HTTP服务器
- 微信公众平台开发入门
- Android开发精品课程【Java核心知识】
- 韦东山嵌入式Linux第一期视频
- Linux高薪入门实战精品（上）
- Qt网络编程实战之HTTP服务器
- JavaScript for Qt Quick(QML)

- epoll使用详解精髓
- epoll使用详解精髓
- epoll使用详解精髓
- epoll使用详解精髓
- epoll使用详解精髓

查看评论

13楼 a185446828 2016-03-16 15:52发表

超级赞，谢谢，学习

12楼 l_Jason 2015-09-27 10:27发表

博主的if else if,,是不是要改为if if ，因为觉得博主的意思是从客户端读取数据后就要发回客户端，如果用if else 的话，本次接收到数据后，再设置ev.data.fd=sockfd;ev.events=EPOLLOUT|EPOLLET;
博主的意思是不是下次轮训的时候，就会检查这个发送标记，从而发送。但是好像这样不可行。。

改为if if的话，在这里设置标记后，并且把上面两行改为events[i].events=EPOLLOUT|EPOLLET;
events[i].data.fd=sockfd;
这样就可以在服务端收到数据后，再发送给客户端了。。

11楼 dreamzme 2014-06-30 00:06发表

应该是用if()... if()而不是if()..else if ()...否则如果是用ET模式，这样就有问题了。

Re: nvfumayx 2015-05-28 17:03发表

回复dreamzme：我认为查看事件的逻辑是 if {} else { if if } 这样，否则对于一个socket，在同一时间段内读写都就绪，并且在et模式下，文章的代码逻辑的会遗漏写就绪这个事件

Re: xiaobendanlei 2014-07-10 10:10发表

回复dreamzme：if if,是不行的，根据上下文就可以看出，if if 下listen之后直接read了，且ET模式，当epoll_wait一次后，是收集当前时刻所有监控fd的情况。如果if if，那么主逻辑里面一直执行下去，混乱了。
整个代码是可以运行的，当是要考虑ET模式下，阻塞read

10楼 liuhuan503 2013-11-18 21:21发表

恩，写的比较详细，但没有运行，不知道bug多不多

9楼 leelin911 2013-08-26 20:11发表

写的很好，受益了不少。不过，好像把所有的工作都堆积到服务器一个进程中了，从时序来讲，服务器负担很重的吧。如果在accept之前服务器主进程先将listen纳入到epoll后，等有连接了，即accept返回后再fork一下，让子进程单独处理客户的请求。并将accept返回值以参数的形式传入到exec函数中，这样服务器子进程再将其纳入到子进程自己的epoll里面。是不是会更好啊？

8楼 江南烟雨 2013-05-28 15:57发表

里面错误很多的好么？首先是函数int epoll_create(int size)的size参数的解释：应该是自从linux2.6.8之后，size参数是被忽略的，只要它比0大就可以。man一下就知道了！

Re: kvl 2013-08-23 17:22发表

很好啊，找个例子挺难的。。

Re: erbort 2013-08-29 13:46发表

回复kvl：有2个API，其中一个是epoll_create1(int _flags)。

Re: 江南烟雨 2013-08-23 20:24发表

回复kvl：英文资料挺多，你可以google搜下英文的~

7楼 zly0706 2013-05-17 04:48发表

大哥 你自己都沒編譯運行 在搞什麼

6楼 思路清晰的小王 2012-07-26 14:30发表

if(events[i].data.fd==listenfd)，为什么等于listenfd的时候是有新连接到来?

Re: eureka_cs 2013-04-13 22:14发表

回复思路清晰的小王：因为前面一定有listenfd = listen(...);这个listenfd也是文件描述符，也加入到epoll里面，如果这个描述符有事件通知，那么肯定是有新的连接过来啦。ps:楼主写的很好，很明白，收藏了。

5楼 FranciscoLin 2011-08-17 12:43发表

很明白 最近正在研究这个

4楼 ziyou飞翔 2011-06-14 08:35发表

不错不错,讲的很好.

3楼 FlyingSnow0311 2011-03-11 16:05发表

就一个线程在操作，可能不可能双工了。你多创建几个线程试试。

2楼 minzhi202 2011-02-22 20:36发表

[e03]这个可能只是测试程序，所以没考虑全双工吧，对于新手来言还是有不错的学习效果的。

1楼 sinservice 2010-03-18 17:14发表

这个服务器似乎有问题。为什么写同时就不可以读，读同时不可以写，socket应该是全双工的。

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved