select的缺陷

2023年5月9日 11:27

① 监听和就绪 耦合

② fd_set 固定大小为1024，不能更改

③ 每次调用 select，把 fd_set 从用户态拷贝到内核态

④ 就绪机制不合理 → 轮询机制不合理

（海量连接，少量就绪）

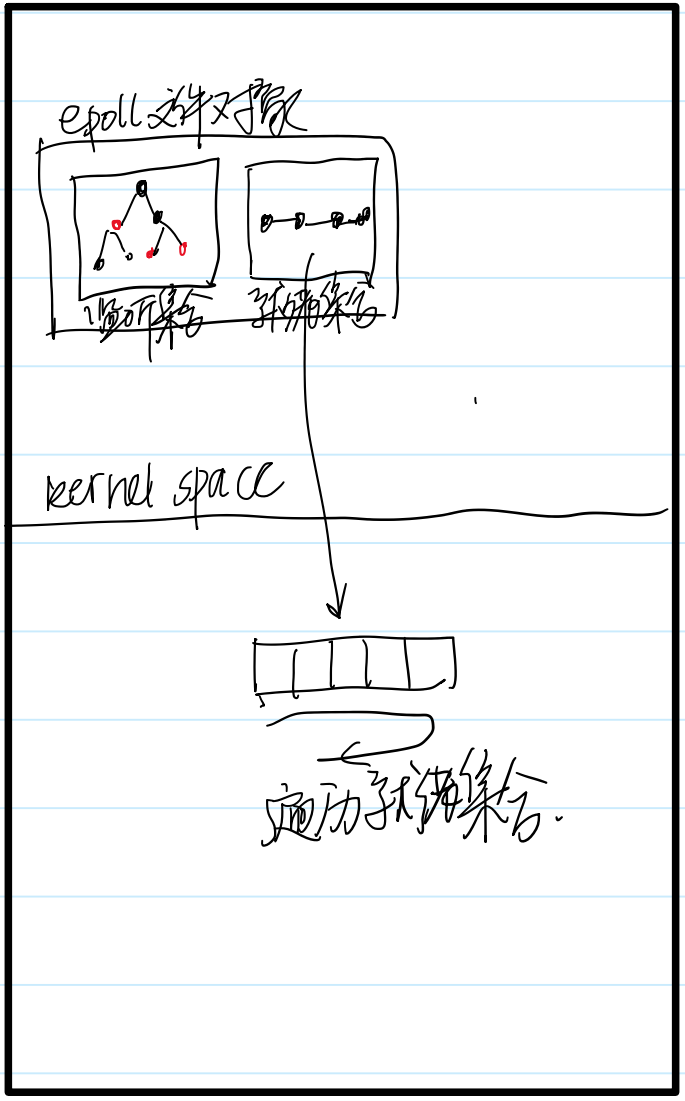
epoll

事件

2023年5月9日 11:40

轮询

IO多路复用 "升级"select 效率高



① Epoll 采用 C 文件对象. (内核区)

② 监听集合采用红黑树, 大小无限制

③ 监听和就绪分离

④ 时间遍历就绪集合

epoll的基本流程

2023年5月9日 14:26

* **epoll_create**(2) creates a new **epoll** instance and returns a file descriptor referring to that instance. (The more recent **epoll_create1**(2) extends the functionality of **epoll_create**(2).)

* Interest in particular file descriptors is then registered via **epoll_ctl**(2), which adds items to the interest list of the **epoll** instance.

* **epoll_wait**(2) waits for I/O events, blocking the calling thread if no events are currently available. (This system call can be thought of as fetching items from the ready list of the **epoll** instance.)

epoll_create 创建epoll文件对象

epoll_ctl 增加监听

epoll_wait 陷入等待

fd_set

FD_ZERO / FD_SET

select

epoll_create

2023年5月9日 14:33

```
int epoll_create(int size);
int epoll_create1(int flags);
```

DESCRIPTION

epoll_create() creates a new epoll(7) instance. Since Linux 2.6.8, the size argument is ignored, but must be greater than zero;

epoll_ctl

2023年5月9日 14:37


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

```
EPOLL_CTL_ADD
        Add fd

EPOLL_CTL_MOD
        Change

EPOLL_CTL_DEL
```

事件。

```
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 */
};
```

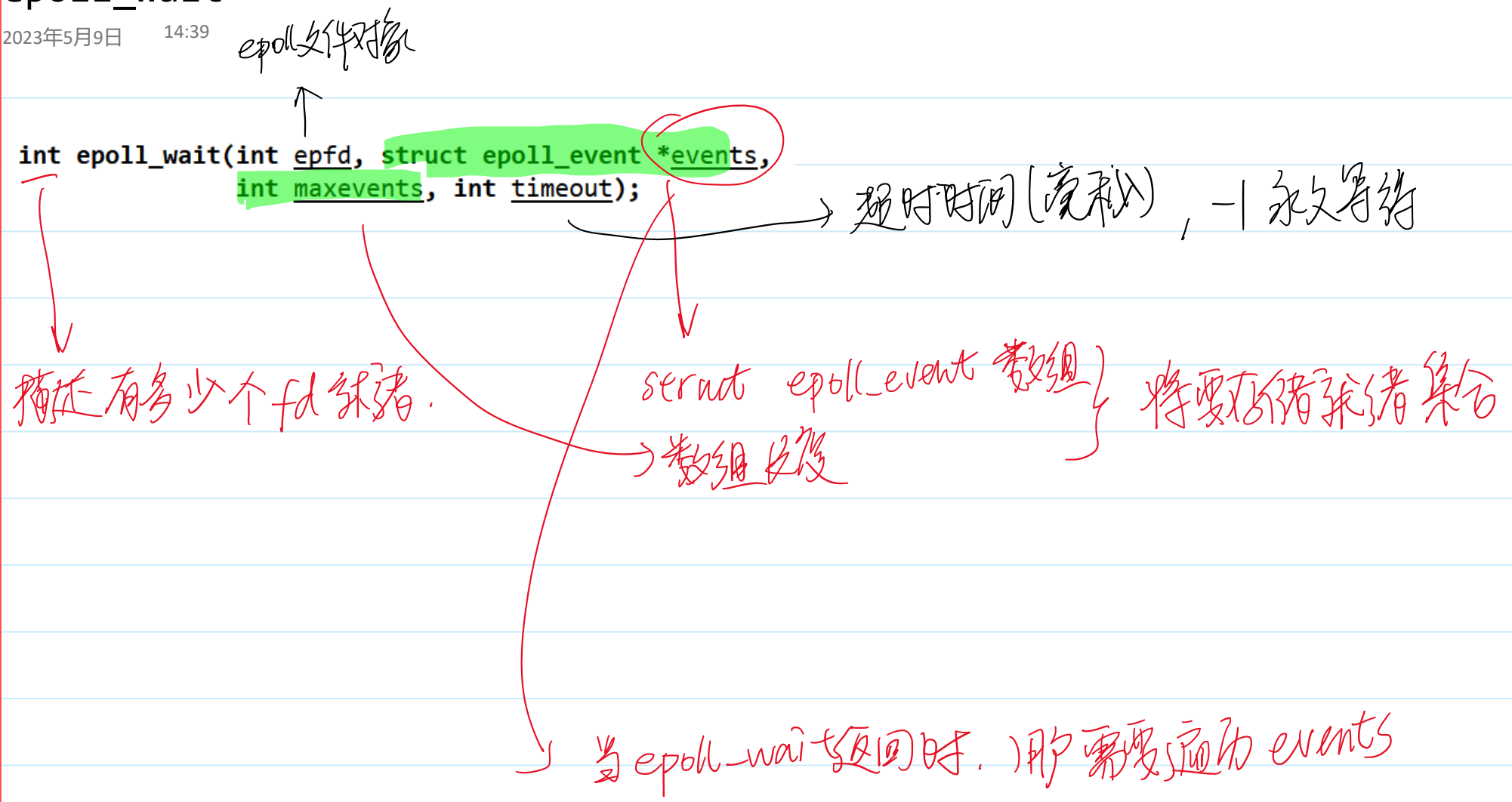
```
EPOLLIN
        Th

EPOLLOUT
```

取fd成员

epoll_wait

2023年5月9日 14:39



用epoll 取代select

2023年5月9日 14:46

```
// 1 fd_set rdset;
int epfd = epoll_create(1); //epoll_create 取代定义fd_set
// 2 设置监听 取代FD_SET
// 如果使用epoll_ctl 可以在循环外面使用
struct epoll_event event;
event.data.fd = STDIN_FILENO;
event.events = EPOLLIN;
epoll_ctl(epfd,EPOLL_CTL_ADD,STDIN_FILENO,&event); // FD_SET
event.data.fd = netfd;
event.events = EPOLLIN;
epoll_ctl(epfd,EPOLL_CTL_ADD,netfd,&event);
char buf[4096];
```

注意事项

2023年5月9日 15:03

```
while(1){
    //FD_ZERO(&rdset);
    //FD_SET(STDIN_FILENO,&rdset);
    //FD_SET(netfd,&rdset);
    //3 select(netfd+1,&rdset,NULL,NULL,NULL);
    struct epoll_event readySet[2];
    int readyNum = epoll_wait(epfd,readySet,2,-1);

 for(int i = 0; i < readyNum; ++i){
     if(readySet[i].data.fd == STDIN_FILENO){
         bzero(buf,sizeof(buf));
         ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
         if(sret == 0){
             send(netfd,"nishigehaoren",13,0);
             goto end;
         }
         send(netfd,buf,sret,0);
     }
     else if(readySet[i].data.fd == netfd){
         bzero(buf,sizeof(buf));
         ssize_t sret = recv(netfd,buf,sizeof(buf),0);
         if(sret == 0){
             printf("hehe\n");
             goto end;
         }
         printf("buf = %s\n", buf);
     }
 }
```

① 创建 epoll_create

② 监听 epoll_ctl
(可以放循环外)

③ 准备好一个 epoll_event 数组
用来保存就绪事件

④ epoll_wait 返回之后，
遍历 events 数组

使用epoll改写作业

2023年5月9日 15:13

非阻塞和阻塞的区别

2023年5月9日 16:11

read 磁盘文件 不会阻塞.

{ read 管道
  read stdin
  read/recv socket } 会阻塞

```
while(1){
    read(磁盘文件)->ret
    ret为0退出
}
```

```
while(1){
    recv(socket)
}
```

把管道弄成非阻塞式

2023年5月9日 16:23

```
int fcntl(int fd, int cmd, ... /* arg */ );
```

F_GETFL (void)
Return (a

F_SETFL (int)

FL file flag

```c
int setnonblock(int fd){
    int flag = fcntl(fd,F_GETFL);//获取已经打开的fd的属性
    flag = flag|O_NONBLOCK;//增加一个非阻塞属性
    int ret =fcntl(fd,F_SETFL,flag);//修改fd的属性
    ERROR_CHECK(ret,-1,"fcntl");
    return 0;
}
int main(int argc, char *argv[])
{
    int fd = open("1.pipe",O_RDONLY);
    setnonblock(fd);
    char buf[1024] = {0};
    while(1){
        bzero(buf,sizeof(buf));
        ssize_t sret = read(fd,buf,3);
        printf("sret = %ld, buf = %s\n", sret, buf);
        sleep(1);
    }
    return 0;
}
```

```
[liao@ubuntu Linuxday_22]$ ./5_read_pipe
sret = -1, buf =
sret = 3, buf = hel
sret = 2, buf = lo
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = -1, buf =
sret = 0, buf =
sret = 0, buf =
sret = 0, buf =
sret = 0, buf =
```

读到数据

没有数据

对端断开

5种IO模型

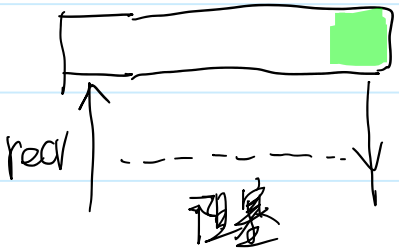
2023年5月9日 16:34

≪Unix网络编程≫

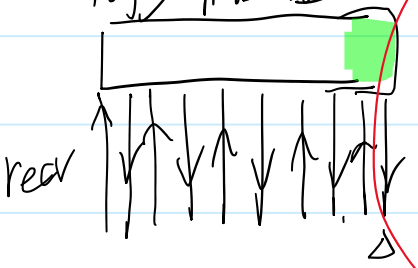
recv
func

while(1){
recv(fd1)
recv(fd2)
recv(fd3)
}
func

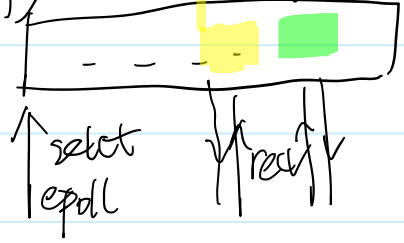
同步阻塞



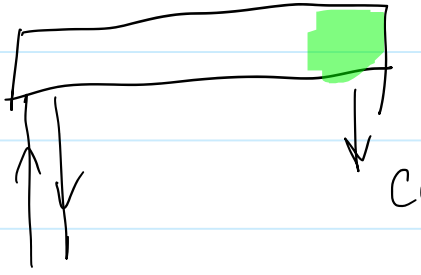
同步非阻塞



同步IO多路复用

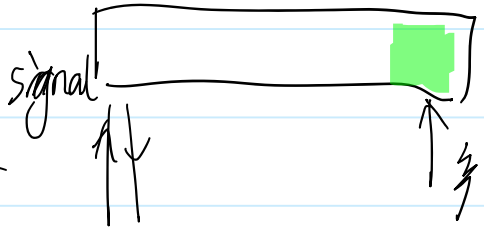


异步IO



aio_read(callback)
func

信号驱动IO

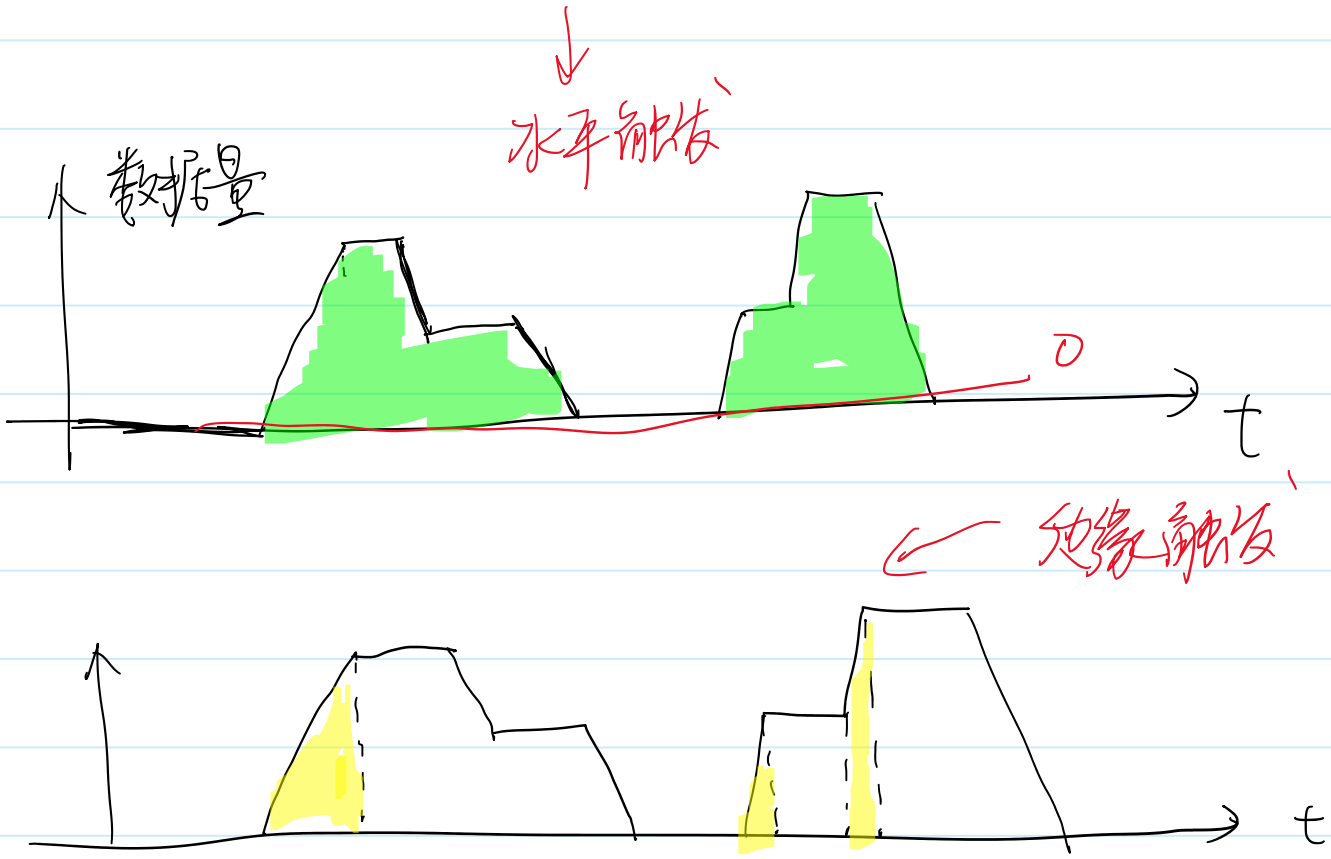


read( )
func

触发方式

2023年5月9日 17:02

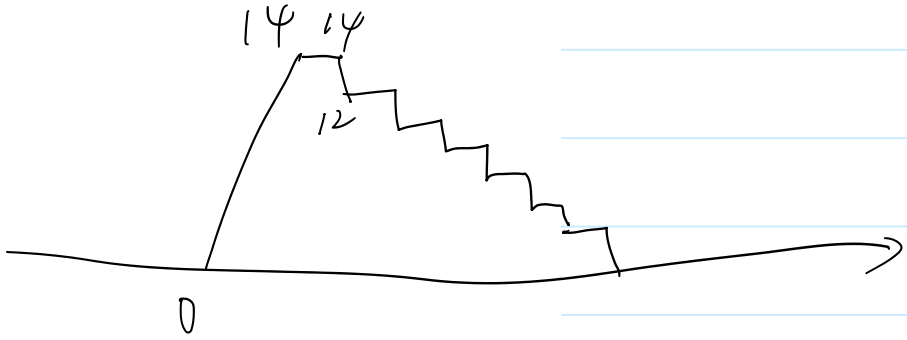
socket 被关闭 or 读缓冲区有数据 ⟶ 唤醒线程.



epoll默认是水平触发

2023年5月9日 17:11

```
[liao@ubuntu Linuxday_22]$ ./6_server_epoll_ET 0.0.0.0 1234
netfd = 4
client ip = 192.168.118.128, port = 49672
epoll_wait ready!
buf = ni
epoll_wait ready!
buf = sh
epoll_wait ready!
buf = ig
epoll_wait ready!
buf = eh
epoll_wait ready!
buf = ao
epoll_wait ready!
buf = re
epoll_wait ready!
buf = n
```



2023年5月9日 17:23

```
event.data.fd = netfd;
event.events = EPOLLIN|EPOLLET;//给socket增加边缘触发属性
epoll_ctl(epfd,EPOLL_CTL_ADD,netfd,&event);
```

```
while(1){
    bzero(buf,sizeof(buf));
    sret = recv(netfd,buf,sizeof(buf)-1,MSG_DONTWAIT);
    printf("sret = %ld, buf = %s\n", sret, buf);
    if(sret == -1){
        break;
    }
    else if(sret == 0){
        printf("hehe\n");
        goto end;
    }
}
```

pool

进程池 线程池

2023年5月9日 17:26

Linux Apache MySQL PHP
LAMP

最传统的服务器框架

```
while(1){
    accept()
    fork/pthread_create
}
```

master-worker 模型 process-based.

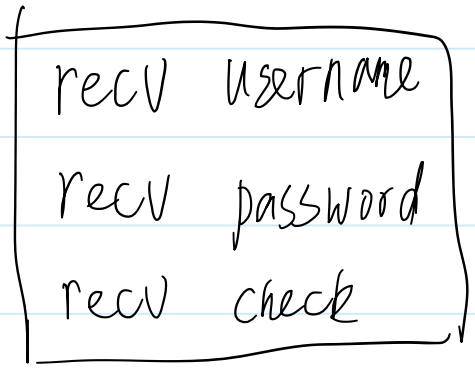
Apache Httpd

一个连接对应一个进程/线程 资源占用大，低并发.

事件驱动框架. IO多路复用 ① 资源小 → 高并发
② 任务调度，用户代码完成.

事件驱动模型的特点

2023年5月9日 17:43



Apache

多线程

```
threadFunc(){
    recv
    recv
    recv
}
```

Nginx

事件驱动

```
epoll_wait();
if(){
    recv
}
if(){
    recv
}
if(){
    recv
}
```

event

进程池的方案

2023年5月9日 17:52