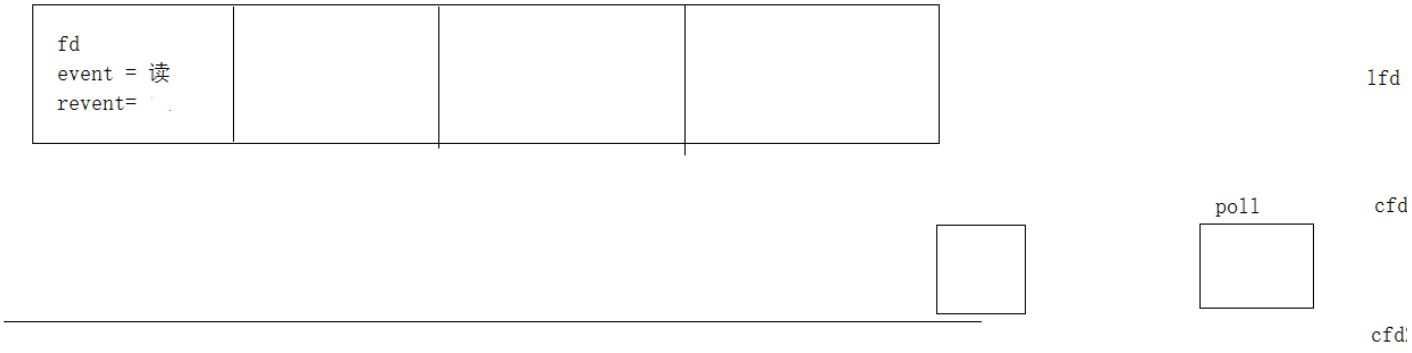
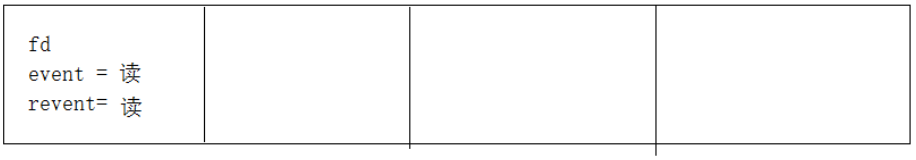
04epoll

1 poll
优点: 相对于select没有最大1024文件描述符限制
请求和返回是分离





2 poll API

```
#include <poll.h>
 int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

功能: 监听多个文件描述符的属性变化
参数:
fds : 监听的数组的首元素地址
nfds: 数组有效元素的最大下标+1
timeout : 超时时间 -1是永久监听 >=0 限时等待

数组元素:
struct pollfd

```
struct pollfd {
   int   fd;         /* file descriptor */ 需要监听的文件描述符
   short events;     /* requested events */需要监听文件描述符什么事件  EPOLLIN 读事件   EPOLLOUT写事件
   short revents;    /* returned events */ 返回监听到的事件    EPOLLIN 读事件   EPOLLOUT写事
};
```

3 poll相对与sellect的优缺点
优点:
没有文件描述符1024的限制
请求和返回是分离的

缺点和select一样:
每次都需要将需要监听的文件描述符从应用层拷贝到内核
每次都需要将数组中的元素遍历一遍才知道那个变化了
大量并发,少量活跃效率低

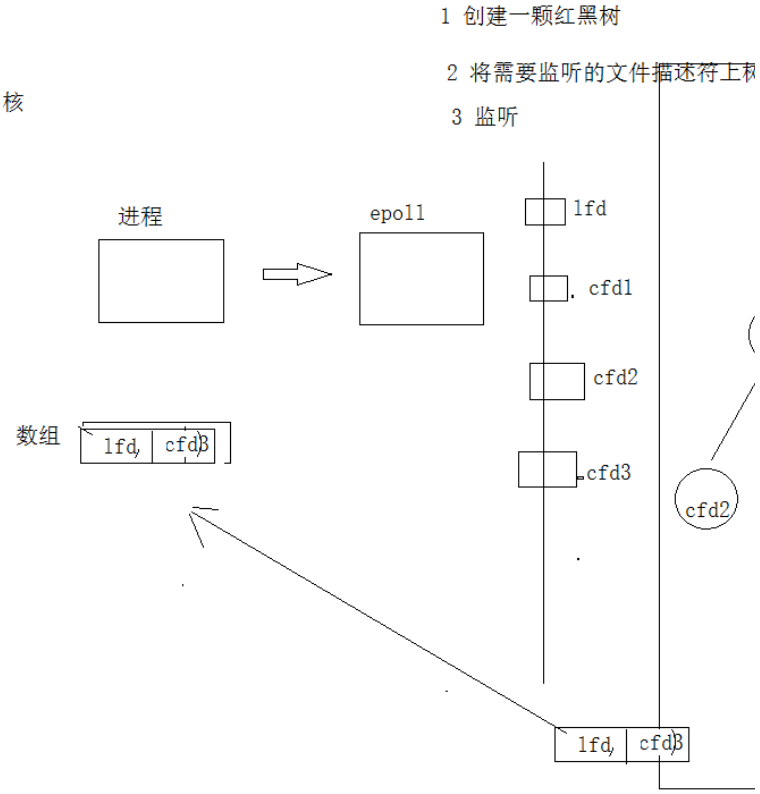
4 epoll的工作原理

1 特点:

没有文件描述符1024的限制

以后每次监听都不要在此将需要监听的文庙描述符拷贝到内核

返回的是已经变化的文件描述符, 不需要遍历树



5 epollAPI

a> 创建红黑树

```
#include <sys/epoll.h>
  int epoll_create(int size);
```

参数:

size : 监听的文件描述符的上限, 2.6版本之后写1即可,

返回: 返回树的句柄

b> 上树 下树 修改节点

epoll_ctl

```
#include <sys/epoll.h>
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
```

参数:

epfd : 树的句柄

op : EPOLL_CTL_ADD 上树 EPOLL_CTL_DEL 下树 EPOLL_CTL_MOD 修改

fd : 上树,下树的文件描述符

event : 上树的节点

```
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 */  需要监听的事件
        epoll_data_t data;        /* User data variable */ 需要监听的文件描述符
    };
```

将cfd上树

```
  int epfd =  epoll_create(1);
struct epoll_event ev;
ev. data.fd = cfd;
ev.events = EPOLLIN;
epoll_ctl(epfd, EPOLL_CTL_ADD,cfd, &ev);
```

c> 监听

```
#include <sys/epoll.h>

   int epoll_wait(int epfd, struct epoll_event *events,
                  int maxevents, int timeout);
```

功能: 监听树上文件描述符的变化

epfd : 数的句柄

events : 接收变化的节点的数组的首地址

maxevents : 数组元素的个数

timeout : -1 永久监听 大于等于0 限时等待

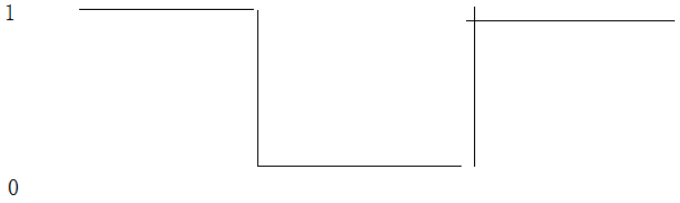
返回值: 返回的是变化的文件描述符个数

6 epoll的工作方式

水平触发 LT

边沿触发 ET

epoll_wait 的水平触发和边沿触发



水平触发:　　持续的高电平 或则 持续的低电平
边沿触发　　电平有高到低的一个变化　或则由低到高的变化

1 监听读缓冲区的变化
水平触发：只要读缓冲区有数据就会触发epoll_wai
边沿触发：数据来一次，epoll_wait只触发一次

2 监听写缓冲区的变化
水平触发：只要可以写，就会触发
边沿触发：数据从有到无，就会触发

因为设置为水平触发,只要缓存区有数据epoll_wait就会被触发,epoll_wait是一个系统调用,尽量少调用
所以尽量使用边沿触发,边沿出触发数据来一次只触发一次,这个时候要求一次性将数据读完,所以while循环读,读到最后read默认带阻塞,不能让read阻塞,因为不能再去监听,
设置cfd为非阻塞,read读到最后一次返回值为-1.判断errno的值为EAGAIN,代表数据读干净

工作中 边沿触发 + 非阻塞 = 高速模式