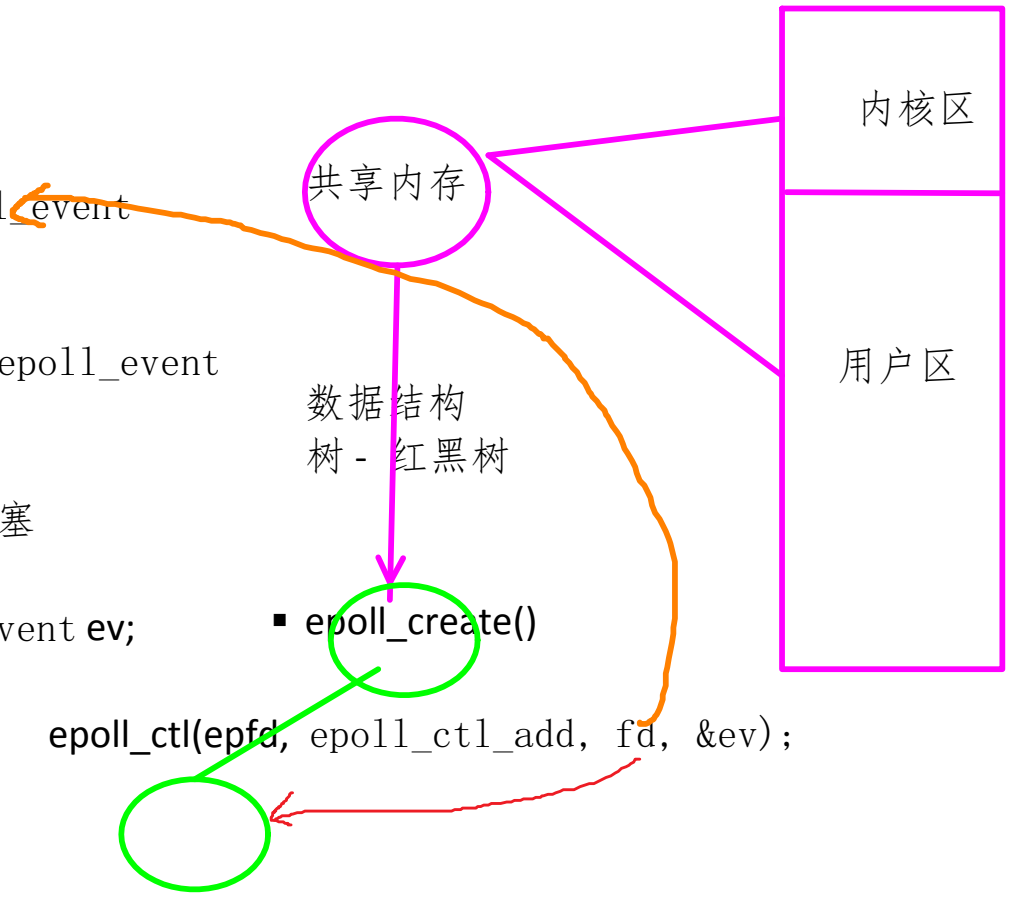
复习

2017年6月5日 9:05

1G内存 = 10万

1. epoll
   - 三个函数
     - epoll_create()
     - epoll_ctl();
       - struct epoll_event
     - epoll_wait();
       - 1 - epfd
       - 2 - struct epoll_event all[];
       - 数组大小
       - 函数是否阻塞
   - 水平 - 默认
     - struct epoll_event ev;
     - ev.events
       - epollin
       - epollout
       - epollerr
   - 边沿
     - ev.events
       - epollin | epollet
       - epollout | epollet
       - epollerr | epollet
   - 边沿非阻塞
     - 设置非阻塞
     - fctnl
     - while((len =recv()) > 0)
     - if(len==0)
       - 客户端断开了连接
       - len == -1
         - 读了没有数据的fd
           - errno == EAGAIN
           - 缓冲区读完了
2. upd
   - 服务器
   - 客户端

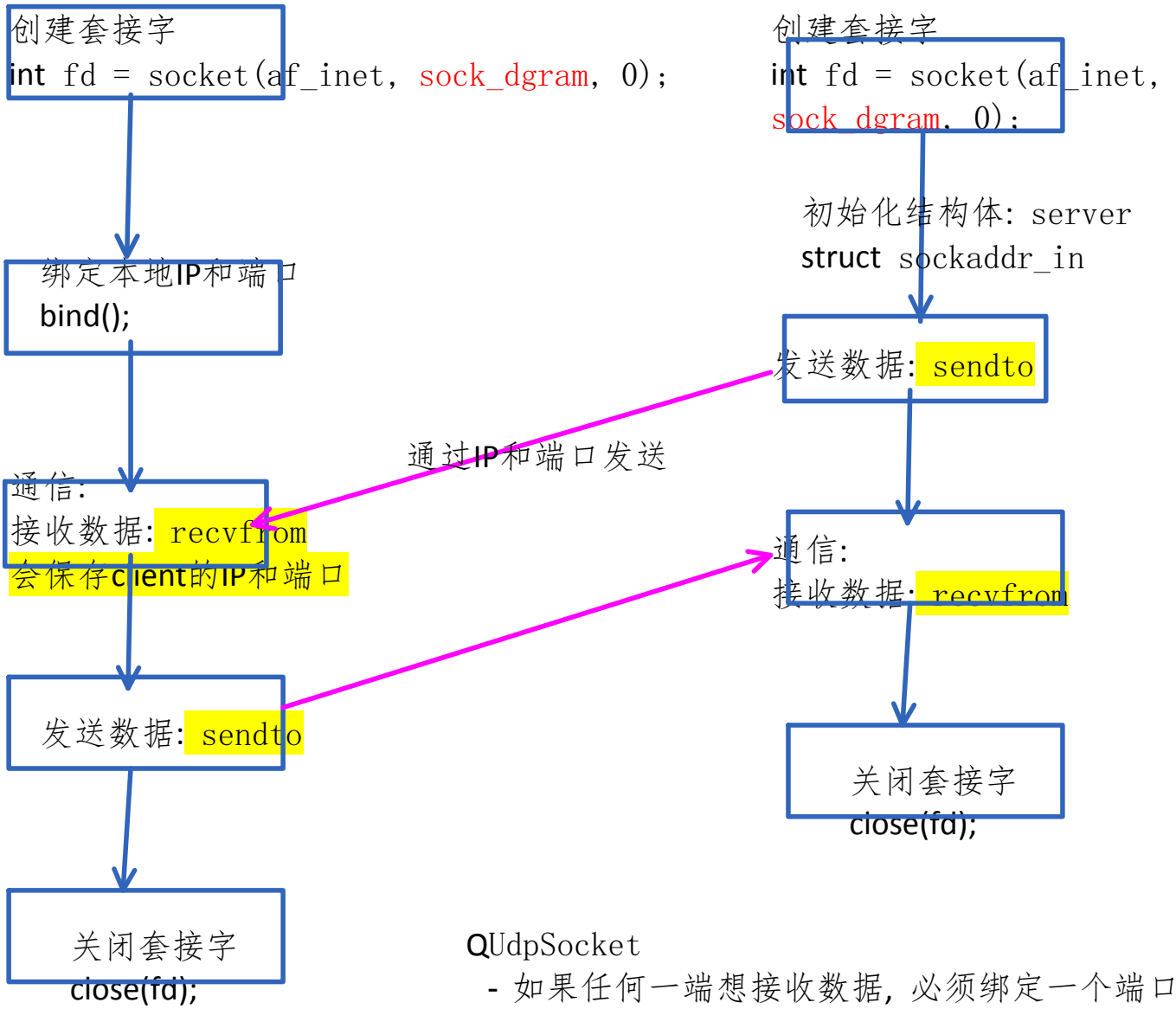


Udp通信流程

2017年6月4日 16:44

server - fd只有一个

客户端 - fd只有一个



QUdpSocket

- 如果任何一端想接收数据, 必须绑定一个端口

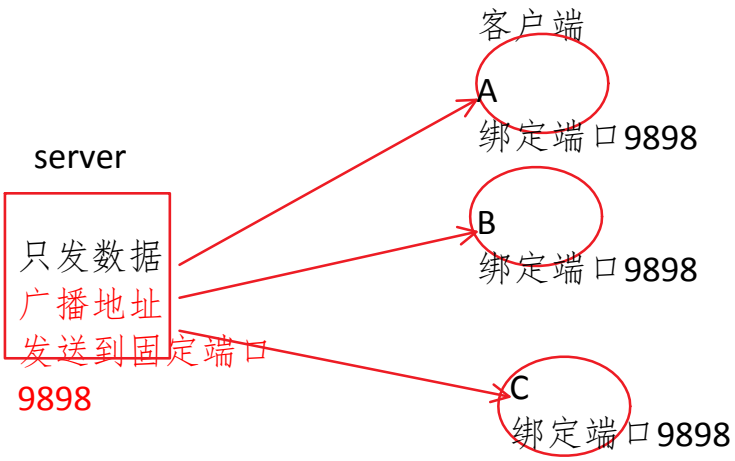
tcp udp使用场景

2017年6月5日 9:33

1. tcp使用场景
    - 对数据安全性要求高的时候
        - 登录数据的传输
        - 文件传输
    - http协议
        - 传输层协议 - tcp
2. udp使用场景
    - 效率高 - 实时性要求比较高
        - 视频聊天
        - 通话
    - 有实力的大公司
        - 使用upd
        - 在应用层自定义协议, 做数据校验

广播

1. 广播
   - 服务器
     - 创建套接字 - socket
     - fd绑定服务器IP和端口
     - 初始化客户端IP和端口信息
       - struct sockaddr_in cli;
       - cli.sin_family = af_inet;
       - cli.port = htons(9898);
       - inet_pton(af_inet, "xxx.xxx.123.255", &cli.adr);
       -);
     - 发送数据
       - sendto(fd, buf, len, 0,)
     - 设置广播权限
       - setsockopt();
   - 客户端
     - 创建套接字
     - 显示绑定IP和端口
       - bind();
     - 接收数据 - server数据
       - recvform();
   - 适用范围
     - 只适用于局域网



广播地址:
xxx.xxx.123.255
255.255. 255. 255

xxx.xxx.123.1 – 网关
xxx.xxx.122.1 – 网关

组播

1. 组播
   - 使用范围：
     - 局域网
     - Internet
   - 结构体

     美人儿去哪

```
struct ip_mreqn
{
        // 组播组的IP地址.
        struct in_addr imr_multiaddr;
        // 本地某一网络设备接口的IP地址。
        struct in_addr imr_interface;
        int   imr_ifindex;  // 网卡编号
};
struct in_addr
{
   in_addr_t s_addr;
};
```

   - 组播地址
     - 224.0.0.0～224.0.0.255

       预留的组播地址（永久组地址），地址224.0.0.0保留不做分配，其它地址供路由协议使用；
     - 224.0.1.0～224.0.1.255

       公用组播地址，可以用于Internet；欲使用需申请。
     - 224.0.2.0～238.255.255.255

       用户可用的组播地址（临时组地址），全网范围内有效；
     - 239.0.0.0～239.255.255.255

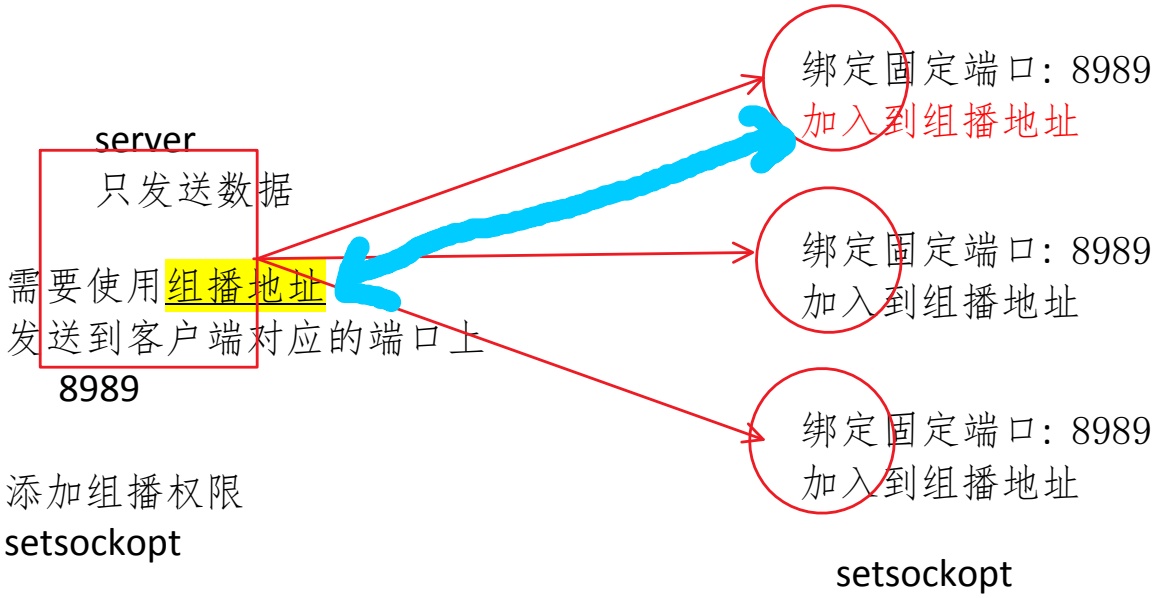
本地管理组播地址，仅在特定的本地范围内有效。

- 服务器端操作:
- 客户端操作:

===组播 - 结构图

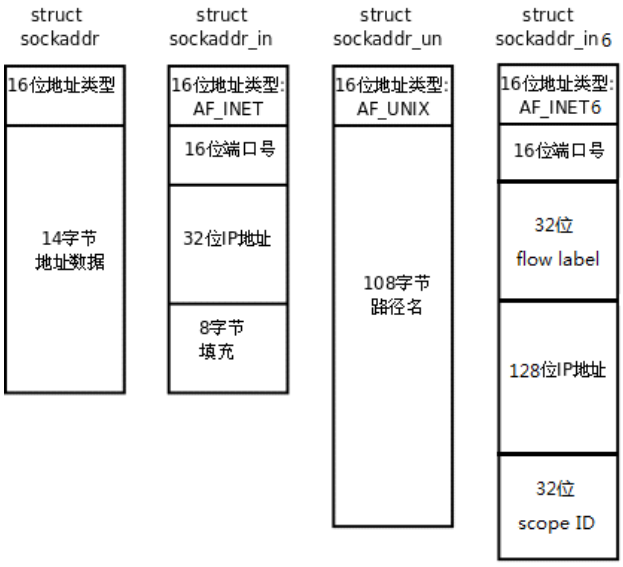
2017年6月5日 11:10

客户端 - 只接收数据，不发送



本地套接字

1. 文件格式:
   - 管道: p
   - 套接字: s
     - 伪文件
2. 服务器端
3. 客户端



头文件: sys/un.h

```
#define UNIX_PATH_MAX 108
struct sockaddr_un {
    __kernel_sa_family_t sun_family;
    char sun_path[UNIX_PATH_MAX];
};
```

1. 服务器端
   - 创建套接字
     - int lfd = socket(AF_LOCAL, sock_stream, 0);
   - 绑定 -

     struct sockaddr_un serv;

     serv.sun_family = af_local;

     strcpy(serv.sun_path, "server.socket"); - -现在还不存在

     bind(lfd, (struct sockaddr8)&serv, len); --- 绑定成功套接字文件被创建
   - 设置监听
     - listen();
   - 等待接收连接请求

     struct sockaddr_un client;

     int len = sizeof(client);

     int cfd = accept(ldf, &client, &len);
   - 通信
     - send
     - recv
   - 断开连接
     - close(cfd);
     - close(lfd);

2. 客户端
   - 创建套接字

     int fd = socket(af_local,sock_stream, 0);
   - 绑定一个套接字文件

     struct sockaddr_un client;

     client.sun_family = af_local;

     strcpy(client.sun_path, "client.socket"); - -现在还不存在

     bind(fd, (struct sockaddr*)&client, len); --- 绑定成功套接

- 连接服务器

  struct sockaddr_un serv;

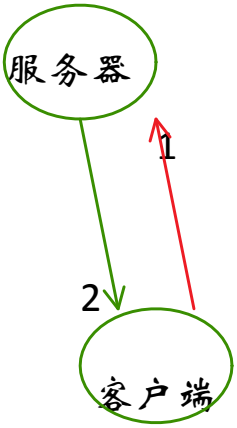
  serv.sun_family = af_local;

  strcpy(serv.sun_path, "server.socket"); - -现在还不存在

  connect(fd, &serv, sizeof(server));
- 通信
  - recv
  - send
- 关闭
  - close

4 - 心跳包

1. 判断客户端和服务器是否处于连接状态
    - 心跳机制
        - 不会携带大量的数据
        - 每个一定时间服务器->客户端/客户端->服务器发送一个数据包
    - 心跳包看成一个协议
        - 应用层协议
    - 判断网络是否断开
        - 有多个连续的心跳包没收到/没有回复
        - 关闭通信的套接字
    - 重连
        - 重新初始套接字
        - 继续发送心跳包

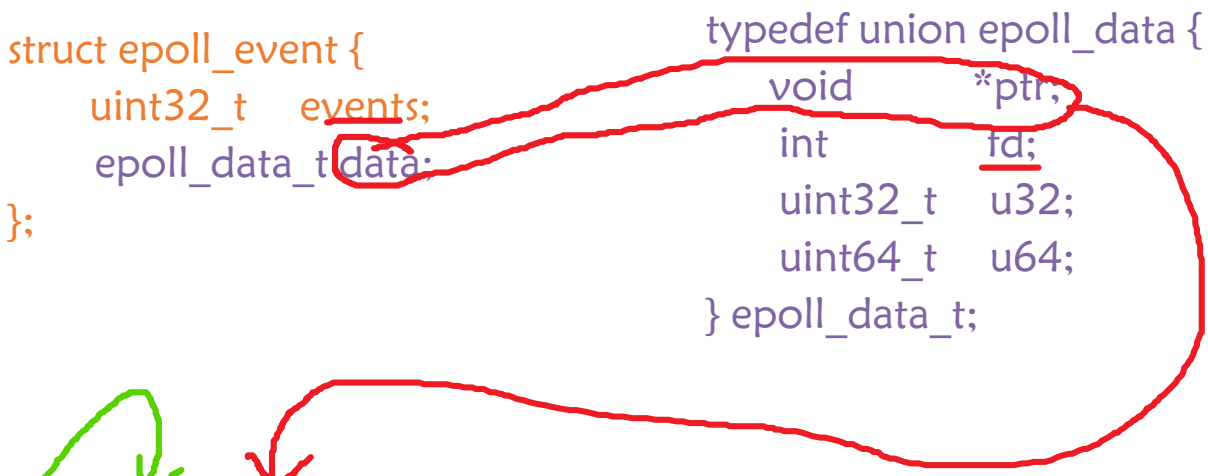


1个字节

--乒乓包

- 比心跳包携带的数据多一些
- 除了知道连接是否存在，还能获取一些信息

- epoll反应堆模型




```
struct myevent_s {
    int fd;          //要监听的文件描述符
    int events;    //对应的监听事件
    void *arg;     //泛型参数
    void (*call_back)(int fd, int events, void *arg);      //回调函数
    int status;         //是否在监听:1->在红黑树上(监听), 0->不在(不监听)
    char buf[BUFLEN];
    int len;
    long last_active;    //记录每次加入红黑树 g_efd 的时间值
};
```

epoll反应堆工作模式

2016年12月15日 11:17

自己的epoll模型

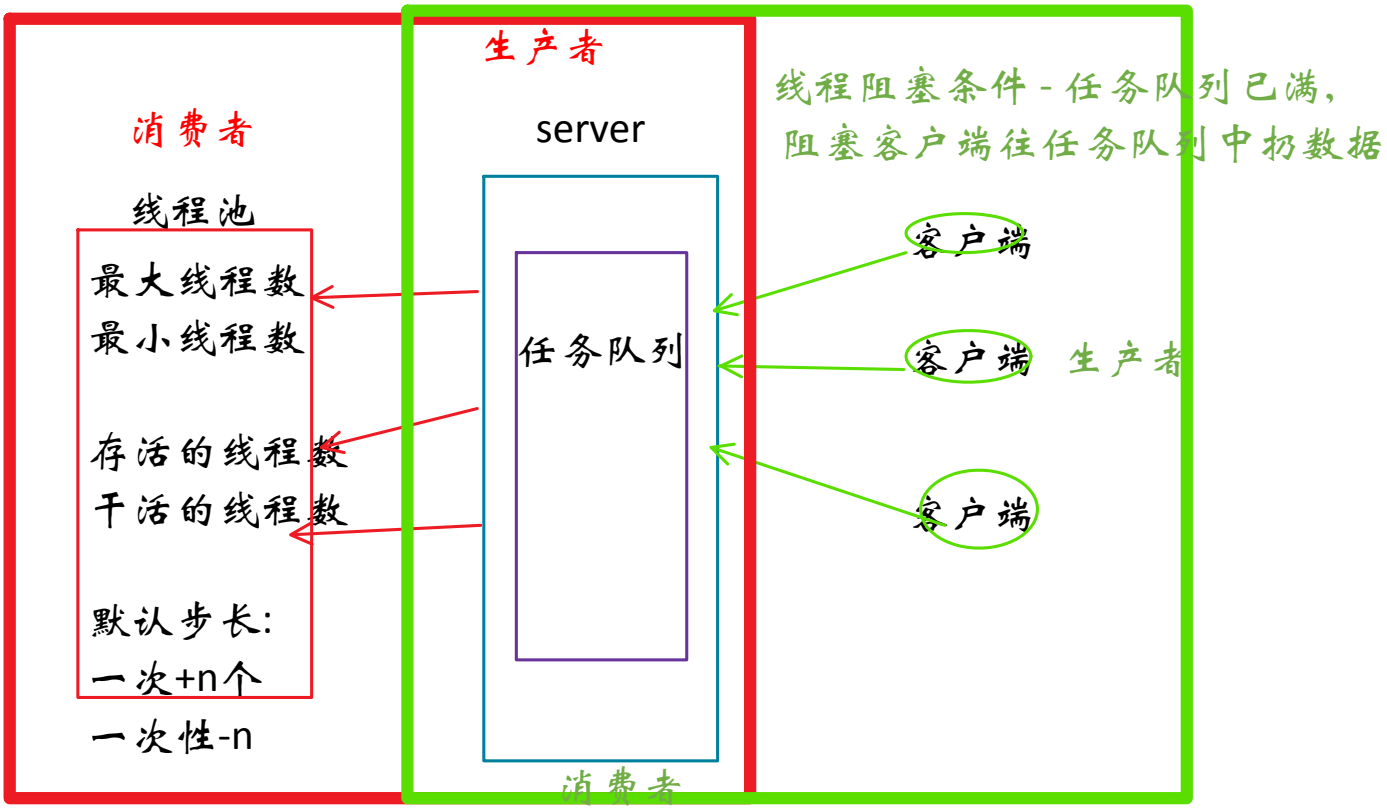
在server - >创建树的根节点-> 在树上添加需要监听的节点
->监听读事件->有返回-> 通信->epoll_wait

在server - >创建树的根节点-> 在树上添加需要监听的节点
->监听读事件->有返回-> 通信(接收数据)->将这个fd从树上删除->监听写事件->写操作->fd从树上摘下来->监听fd的读事件->epoll_wait

EPOllOUT

- 水平模式:
  - struct epoll_event ev;
    - ev.events = EPLLOUT;
    - epoll_wait会一直返回,缓冲区能写数据,该函数会返回,缓冲区满的时候,不返回
- 边缘模式:
  - 第一次设置的时候epoll_wait会返回一次
  - 缓冲区从满->到不满的时候

- 线程池



线程阻塞条件:

任务对列如果为空 cond_empty

pthread_cond_wait(&cond_empty, &mutex);

任务队列中有数据:

激活阻塞在条件变量上的线程:

pthread_cond_signal(&cond_empty);

pthead_cond_broadcast(&cond_empty);

1. 初始化一些线程
2. 需要有一个管理者线程
   a. 如果使用率超过一定的百分比
      i. 创建线程:按照一定的步长增长
   b. 空闲的线程增多
      i. 销毁线程
         1) 留下的比实际多一些
3. 线程工作的时候:
   a. 处理数据的时候:
      i. 互斥锁

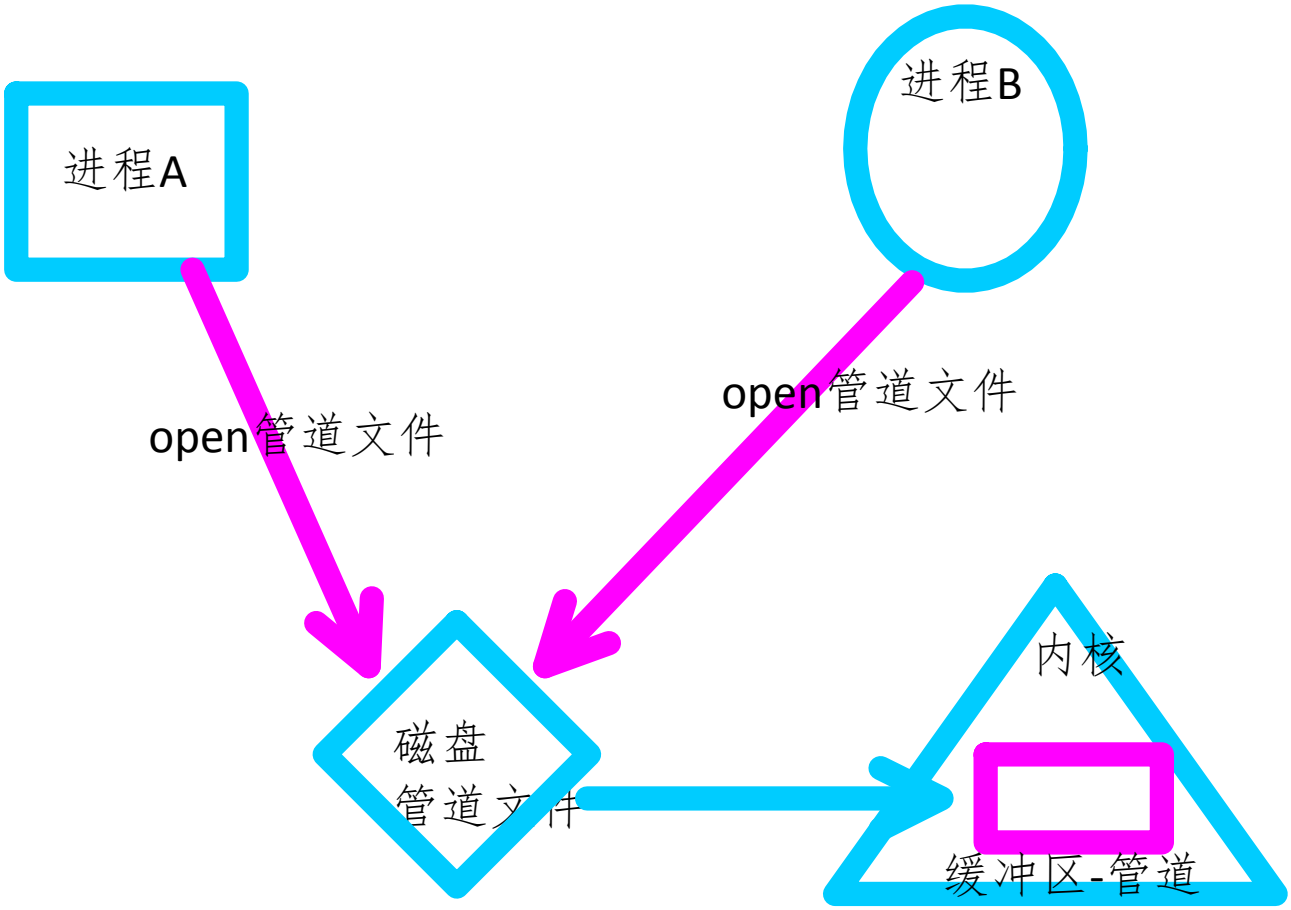
ii. 条件变量

管理者线程:
- 计算线程不够用
  - 创建线程
- 空闲线程太多
  - 销毁
    - 更新要销毁的线程个数
    - 通过条件变量完成的
      - 如果空闲太多,任务不够
        - 线程阻塞在该条件变量上
      - 发送信号
        - pthread_cond_signal

线程池中的线程:
- 从任务队列中取数据
  - 任务队列任务--
  - 执行任务
- 销毁空闲的线程
  - 让线程执行pthread_exit
  - 阻塞空闲的线程收到信号
    - 解除阻塞
    - 只有一个往下执行
    - 在执行任务之前做了销毁操作
      - 自行退出

使用有名管道进程没有血缘关系的进程间通信

2017年6月5日　　15:04



本地套接字进程通信

2017年6月5日　　15:23