学习目标

1. 熟练掌握三次握手建立连接过程
2. 熟练掌握四次挥手断开连接过程
3. 掌握滑动窗口概念 nginx
4. 掌握错误处理函数封装
5. 实现多进程并发服务器
6. 实现多线程并发服务器

复习

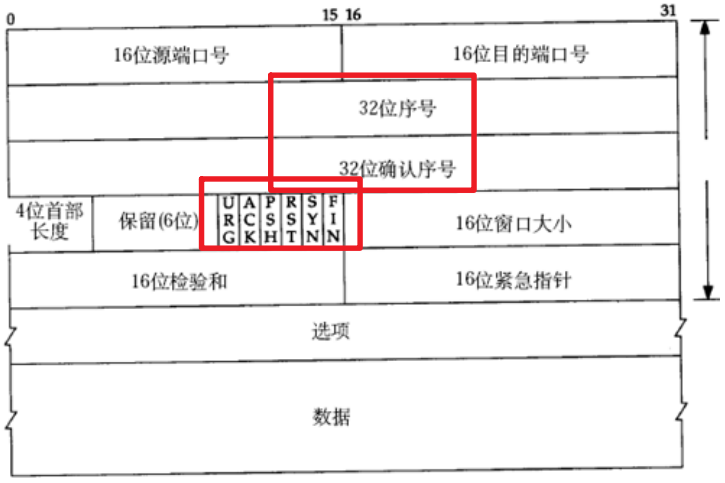
1. 网络开发设计模式
   - c/s
     - 提供桌面客户端
   - b/s
     - 跨平台
     - http
2. 分层模型
   - 七层: 物数网传会表应
   - 四层:
     - 网络接口层
     - 网络层
     - 传输层- udp, tcp
     - 应用层
3. tcp, udp
4. socket
   - 文件(内核的缓冲区)操作
   - socket tcp server
     - 创建套接字
       - int lfd = socket
     - 绑定本地IP和端口
       - struct sockaddr_in serv;
       - serv.port = htons(port);
       - serv. IP= htonl(INADDR_ANY);
       - bind(lfd, &serv, sizeof(serv));
     - 监听
       - listen(lfd, 128);
     - 等待并接收连接请求

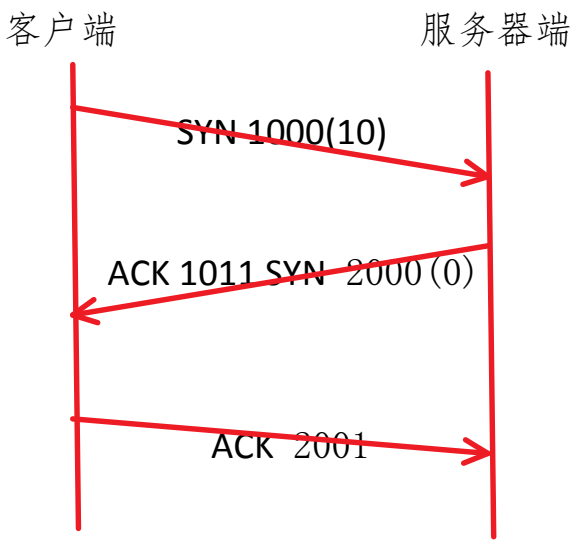
        - struct sockaddr_in client;
        - int len = sizeof(client);
        - int cfd = accept(lfd, &client, &len);
            - cfd - 用于通信的
    - 通信
        - 接收数据: read/recv
        - 发送数据: write/send
    - 关闭:
        - close(lfd);
        - close(cfd);
- 客户端:
    - 创建套接字
        - int fd = socket
    - 连接服务器
        - struct sockaddr_in server;
        - server.port
        - server.ip = (int)  ?????
        - server.family
        - connect(fd, &server, sizeof(server));
    - 通信
        - 接收数据: read/recv
        - 发送数据: write/send
    - 断开连接
        - close(fd);

# 1-tcp客户端编程

## 2-tcp三次握手

### tcp - 面向连接的安全的流式传输



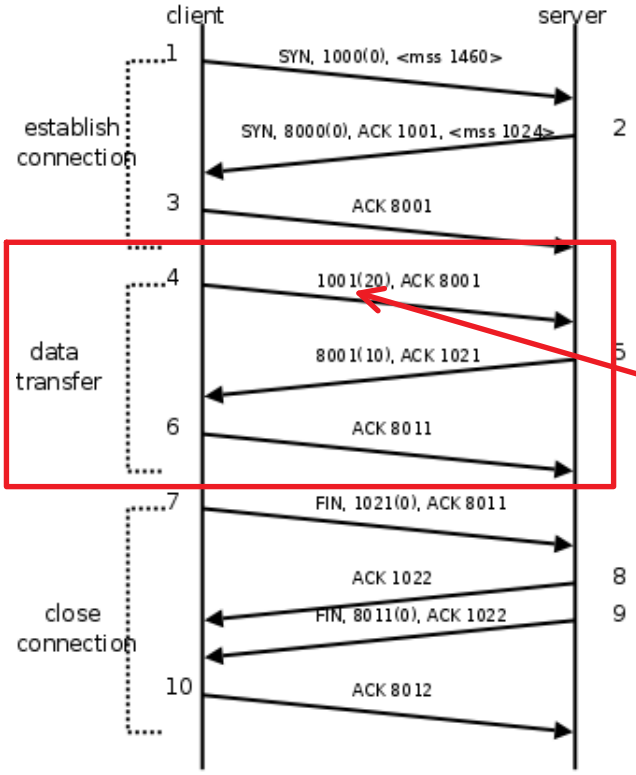




- 标志位:
  - SYN: 请求建立连接
  - ACK: 应答
  - FIN: 断开连接
- 连接需要三次握手:
  - 第一次握手:
    - 客户端
      - 携带标志位: SYN
      - 随机产生32为序号: 1000
        - 可以携带数据()
    - 服务器:
      - 检测SYN值是否为1
  - 第二次握手:
    - 服务器:
      - ACK 标志位 + 确认序号
        - 客户端随机序号+1
      - 发起一个连接请求
        - SYN+32随机序号
          - 2000
    - 客户端:
      - 检测标志位: 1
      - 校验: 确认序号是否正确
  - 第三次握手:
    - 客户端:
      - 发送确认数据包
        - ACK+确认需要
          - 服务器的随机序号+1

- 服务器:
  - 检测: ACK是否为1
  - 检验: 确认序号是否正确

--- tcp连接和数据传输过程

2017年6月1日 11:01




mss – 最大数据长度

客户端:
- connect

服务器:
- accept

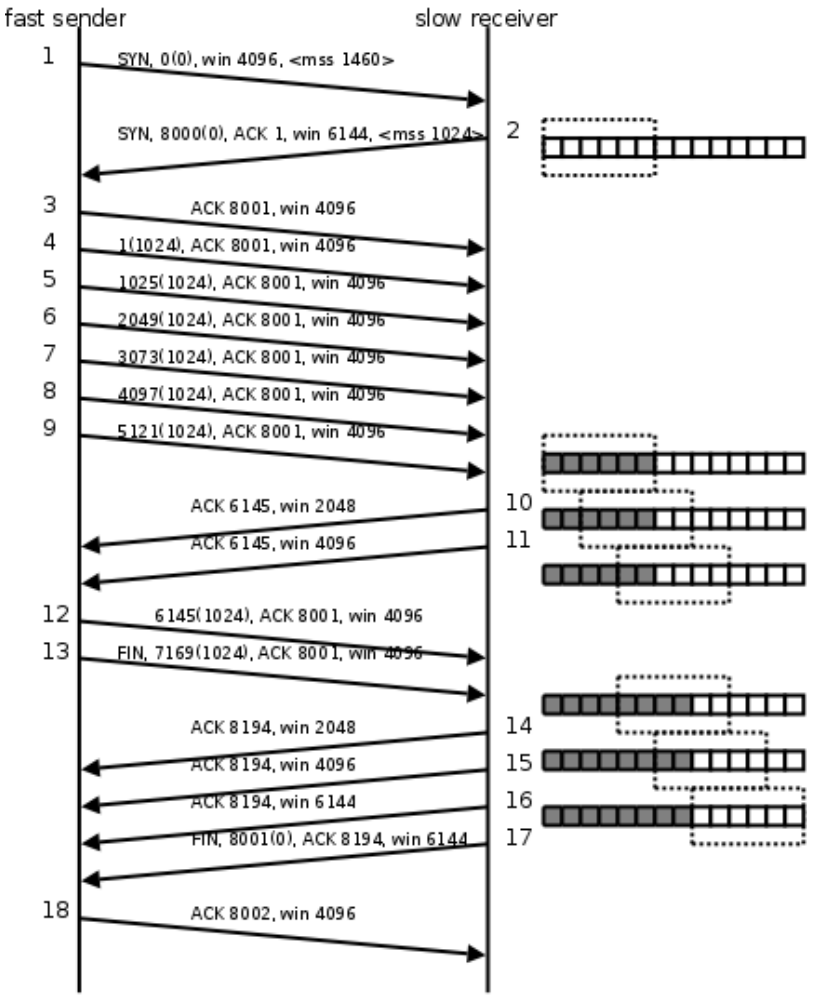
编号:
- 对方最后发送ACK的时候携带的确认序号

断开连接:
- 服务器:
  - close
- 客户端:
  - close

# tcp-滑动窗口

2017年6月1日 11:32

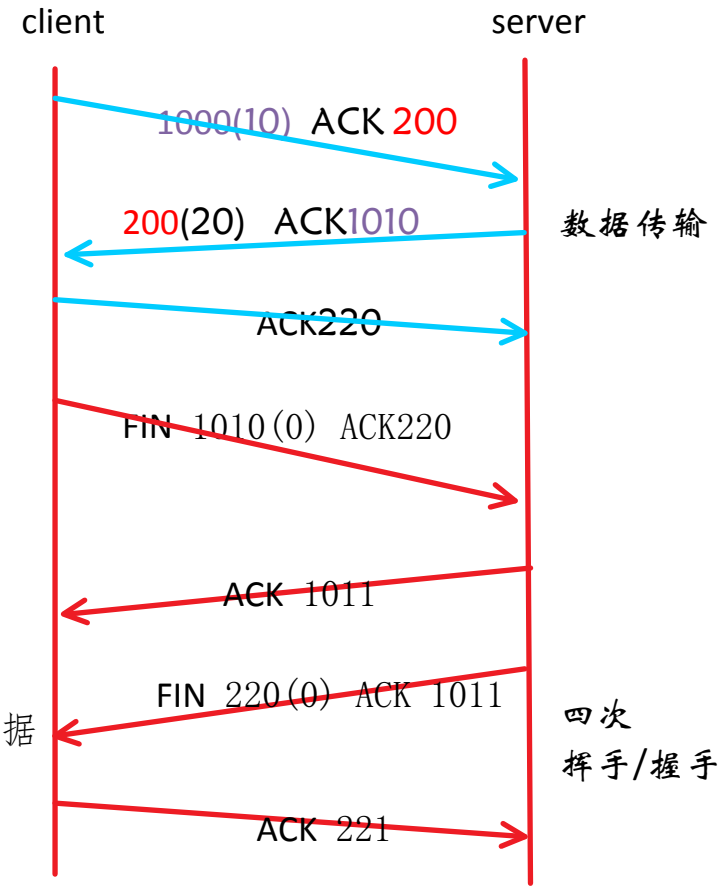
客户端 服务器




1. 滑动窗口
   - 缓冲区
2. win 4096
   - win - 滑动窗口
   - 滑动窗口对应的缓冲区大小

## 3-tcp的四次挥手

1. 哪一端主动断开连接都可以
2. 需要一个标志位: FIN
   编号:
   - 对方最后发送ACK的时候携带的确认序号



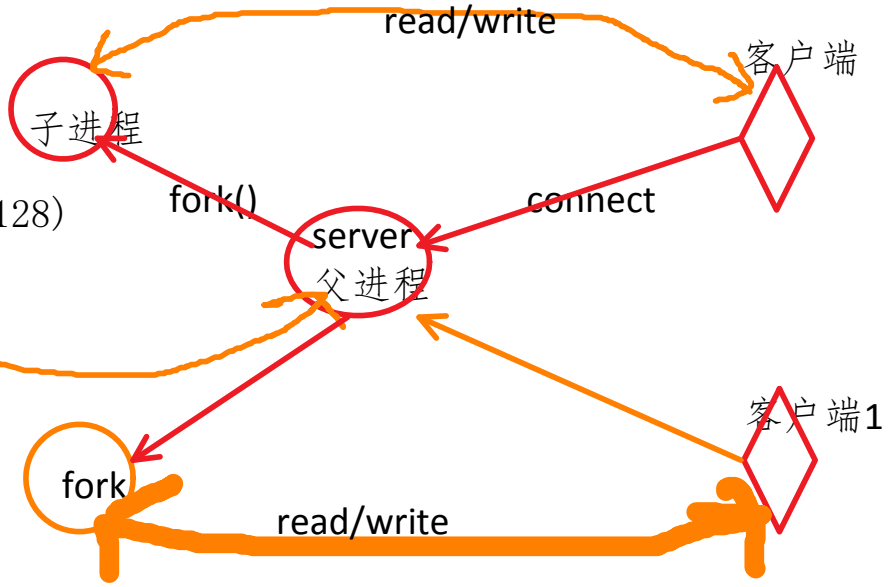

第一次挥手:
- 客户端:
  - 发送断开连接的请求
    - FIN + 序号
    - ACK + 序号
  - server:
    - 检测FIN值是否为1
    - ack的作用告诉对方之前发的数据收到了多少

第二次挥手:
- 服务器:
  - 给client确认数据包
    - ACK + 确认编号
      - FIN对应的序号+1+携带数据大小
- 客户端:
  - 检测: ack值
  - 检测确认序号

第三次挥手:
- 服务器端:
  - 发送断开连接的请求:
    - FIN+序号
    - ACK + 序号
- 客户端:
  - 数据检测

## 4-tcp多进程并发服务器

1. 只能处理单链接
   - 创建套接字 - 监听
   - 绑定
   - 监听 - listen(lfd, 128)
   - ----------
   - 接受连接请求
   - 通信




使用多进程的方式,解决服务器处理多连接的问题:

1. 共享
   - 读时共享,写时复制
   - 文件描述符
   - 内存映射区 -- mmap
2. 父进程 的角色是什么?
   - 等待接受客户端连接 -- accept
     - 有链接:
       - 创建一个子进程 fork()
     - 将通信的文件描述符关闭
3. 子进程的角色是什么?
   - 通信
     - 使用accept返回值 - fd
   - 关掉监听的文件描述符
     - 浪费资源
4. 创建的进程的个数有限制吗?
   - 受硬件限制
   - 文件描述符默认也是有上限的1024
5. 子进程资源回收
   - wait/waitpid
   - 使用信号回收
     - 信号捕捉:
       - signal
       - sigaction - 推荐

- 捕捉信号: SIGCHLD

==== 多进程伪代码

2017年6月1日 15:13

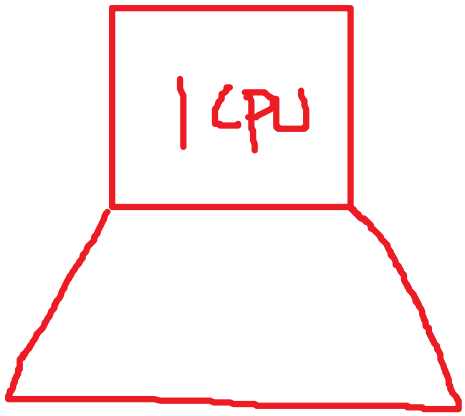
```
void recyle(int num)
{
    while(waitpid(-1, NULL, wnohang) > 0);
}

int main()
{
    // 监听
    int lfd = sock();
    // 绑定
    bind();
    // 设置监听
    listen();

    // 信号回收子进程
    struct sigaction act;
    act.sa_handler = recyle;
    act.sa_falgs = 0;
    sigemptyset(&act.sa_mask);
    sigaction(SIGCHLD, &act, NULL);

    // 父进程
    while(1)
    {
        int cfd = accept();
        // 创建子进程
```

```
pid_t pid = fork();
// 子进程
if(pid == 0)
{
    close(lfd);
    // 通信
    while(1)
    {
        int len = read();
        if(len == -1)
        {
            exit(1);
        }
        else if(len == 0)
        {
            close(cfd);
            break;
        }
        else
        {
            write();
        }
    }
    // 退出子进程
    return 0; //exit(1);
}
else
{
    // 父进程
    close(cfd);
```

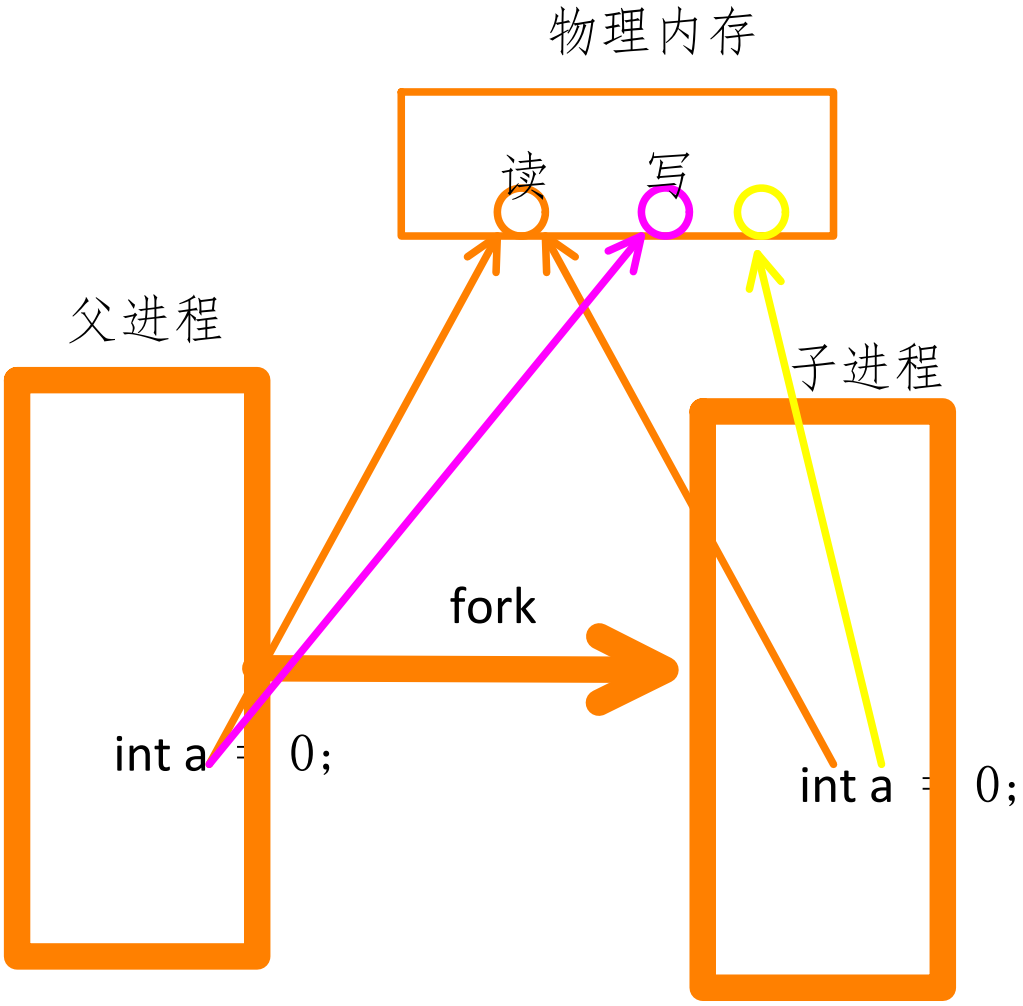
```
            }
        }
    }
```

# 5-tcp多线程并发服务器

并发




2017年6月1日 15:00




a = 8;