# malloc和free

## 前言

## memcpy函数

### 实现memcpy的功能

char\* tempdest=(char\*)dest;

const char\* tempsrc=(const char\*)src;

while (num--) {

\*tempdest=\*tempsrc;

tempsrc=tempsrc+1;

tempdest=tempdest+1;

}

自己实现内存拷贝的功能,很容易忽略到内存重叠的情况,这也是为什么莫名其妙会发生报错的原因。

1. src和dest所指内存区域不能重叠。
2. 与strcpy相比，memcpy并不是遇到'\0'就结束，而是一定会拷贝完n个字节。

3.memcpy用来做内存拷贝，你可以拿它拷贝任何数据类型的对象，可以指定拷贝的数据长度。

## 成员在结构体中的内存偏移量

(struct\_type \*) 0

把一段地址看作struct\_type；

(struct\_type \*) 0)->member

指向它的member成员

&((struct\_type \*) 0)->member

取这个成员的地址

(size\_t) &((struct\_type \*) 0)->member

把地址硬转为size\_t类型

## TCP/IP协议

SYN：发起新连接，当SYN=1，表示这是一个请求建立连接的报文段；当SYN=1，ACK=1时，表示对方同意建立连接，只有在前两次握手中 SYN 才为 1

ACK：确认序号标志，ACK=1 时表示有效。只有当 ACK=1 时，前面的确认号字段才有效，TCP连接建立后，ACK 必须为 1

### 三次握手

1. 客户端向服务端发送一段TCP报文：  
   标志位为SYN=1，表示“请求建立新连接”；序号为seq=x（x通常为1）；随后客户端进入SYN-SENT状态并停止向服务端发送数据
2. 服务端接收到来自客户端的TCP报文之后，结束LISTEN状态，并返回一段TCP报文：  
   标志位为SYN=1，ACK=1，表示“已收到请求并同意创建新连接”；序号为seq=y；确认号为ack=x+1，表示将来自客户端的报文序号seq值加1作为自己确认号ack的值；随后服务端进入SYN-RCVD状态
3. 客户端收到服务端的同意连接TCP报文后，确认了双方间数据传输是正常的，结束SYN-SENT状态，并返回最后一段TCP报文：  
   标志位为ACK=1，表示“已收到同意连接的信号”；序号为seq=x+1，表示将来自服务端的确认号ack值作为自己的序号值；确认号为ack=y+1，表示将来自服务端序号seq值加1作为自己的确认号ack的值；随后客户端进入ESTABLISHED状态  
   服务端收到来自客户端的确认连接TCP报文之后，也确认了双方间数据传输正常，结束SYN-RCVD状态，进入ESTABLISHED状态

[https://blog.csdn.net/Q0717168/article/details/111933810](https://blog.csdn.net/Q0717168/article/details/111933810" \t "https://blog.csdn.net/Q0717168/article/details/_blank)

在第三次握手时就可以携带数据了，因为能够发出第三次握手报文的客户端，肯定接收到来自服务端的第二次握手报文，而伪造IP的客户端是不会接收到第二次报文的，所以，能够发出第三次握手报文的就是合法的用户，服务端在接收到第三次握手的瞬间，状态就会切换为ESTABLISHED，里面携带的数据就可以按照正常流程走了

为何两次握手不可以呢？原因是为了防止服务端开启一些无用的连接增加服务器开销以及防止已失效的连接请求报文段突然又传送到了服务端而产生错误。“第三次握手”是客户端向服务端发送数据，目的是告诉服务器，客户端有没有收到服务器“第二次握手”时传过去的数据，若发送的是“收到了”的信号，服务端接收后就正常建立TCP连接，否则建立TCP连接失败，服务器关闭连接端口，由此减少服务器开销及防止接收到失效请求而导致错误

### 四次挥手

挥手之前主动释放连接的客户端结束ESTABLISHED状态，随后开始“四次挥手”：

1. 客户端向服务端发送TCP报文：  
   标志位为FIN=1，表示“请求释放连接”，序列号为seq=u，随后客户端进入FIN-WAIT-1状态并停止向服务端发送数据
2. 服务端收到从客户端发出的释放连接的TCP报文后结束ESTABLISHED状态，进入CLOSE-WAIT状态并返回一段TCP报文：  
   标志位为ACK=1，表示“收到释放请求”，序列号为seq=v，确认号ack=u+1，表示将来自客户端报文序号seq值加1作为自己的确认号ack的值，随后服务端进入CLOSE-WAIT状态
3. 客户端收到服务端确认结果后，进入FIN-WAIT-2状态。服务端做好了释放连接准备后再次向客户端发出一段TCP报文：  
   标志位为FIN=1，ACK=1，表示“已准备好释放”，序号为seq=w，确认号ack=u+1，服务器进入LAST-ACK状态并停止向客户端发送数据
4. 客户端收到回复后，结束FIN-WAIT-2状态，进入TIME-WAIT状态，并向服务端发送一段报文：  
   标志位为ACK=1，表示“已收到准备释放信号”，序列号为seq=u+1，确认号为ack=w+1，表示将来自服务端报文序号seq值加1作为自己的确认号ack的值，客户端进入TIME-WAIT状态，客户端经过2MSL后进入CLOSE状态；服务器收到确认后，立刻进入CLOSE状态

（MSL指的是Maximum Segment Lifetime：一段TCP报文在传输过程中的最大生命周期。2MSL即是服务端发出为FIN报文和客户端发出的ACK确认报文所能保持有效的最大时长）

## 派生函数重写加不加override的区别

在C++中，关键字`override`用于声明一个成员函数是在派生类中对基类的虚函数进行重写。使用`override`关键字可以帮助编译器在编译时检查重写的正确性。

1. 如果在派生类的成员函数声明中使用`override`关键字来标记对基类的虚函数的重写，编译器会检查是否有相应的基类函数存在，并且具有相同的函数签名。如果没有找到对应的基类函数或者函数签名不匹配，编译器会报错。
2. 如果不使用`override`关键字，编译器仍然会将函数定义为虚函数，并且可以在派生类中对其进行重写，但是编译器不会进行重写的正确性检查。这意味着如果派生类的成员函数的函数签名与基类的虚函数不匹配，编译器也不会发出警告或错误。

总结起来，使用`override`关键字可以提高代码的可读性和维护性，同时也可以帮助捕获一些潜在的错误。

## 总结

以上就是今天要讲的内容，后续会有更多内容。

## 参考资料

版权声明：本文参考了其他资料和CSDN博主的文章，遵循CC 4.0 BY-SA版权协议，现附上原文出处链接及本声明。

1. [https://blog.csdn.net/weixin\_67596609/article/details/130722926](https://blog.csdn.net/weixin_67596609/article/details/130722926" \t "https://blog.csdn.net/weixin_67596609/article/details/_blank)