# 面试学习笔记

## 线程与进程的区别和联系?

线程是否具有相同的堆栈? dll是否有独立的堆栈?

答案：进程是死的，只是一些资源的集合，真正的程序执行都是线程来完成的，程序启动的时候操作系统就帮你创建了一个主线程。每个线程有自己的堆栈。 DLL中有没有独立的堆栈，这个问题不好回答，或者说这个问题本身是否有问题。因为DLL中的代码是被某些线程所执行，只有线程拥有堆栈，如果DLL中的代码是EXE中的线程所调用，那么这个时候是不是说这个DLL没有自己独立的堆栈？如果DLL中的代码是由DLL自己创建的线程所执行，那么是不是说DLL有独立的堆栈？以上讲的是堆栈，如果对于堆来说，每个DLL有自己的堆，所以如果是从DLL中动态分配的内存，最好是从DLL中删除，如果你从DLL中分配内存，然后在EXE中，或者另外一个DLL中删除，很有可能导致程序崩溃

一.进程和线程基本概念

1.进程：进程是操作系统中资源分配的基本单位。进程是操作系统对正在运行程序的一种抽象，可以将进程看作程序的一次运行。

2.线程：线程是操作系统中调度执行的基本单位。一个线程是一个“执行流”，每个线程之间都可以按照顺序执行自己的代码，多个线程“同时”执行多份代码。

二.进程和线程之间的区别与联系

1.一个进程可以包含多个线程，线程在进程的内部。

2.进程之间的资源是独立的，线程之间的资源则是共享的。

每个进程都有独立的虚拟地址空间，也有之间独立的文件描述符表，同一进程的多个线程之间则共用这一份虚拟地址空间和文件描述符表。

3.进程是操作系统中资源分配的基本单位，线程是操作系统中调度执行的基本单位。

4.多个进程同时执行时，如果一个进程崩溃，一般不会影响其他进程，而同一进程内的多个线程之间，如果一个线程崩溃，很可能使得整个进程崩溃。

5.进程的上下文切换速度比较慢，而线程的上下文切换速度比较快。

6.进程的创建/销毁/调度开销大，线程的创建/销毁/调度开销相对少很多。

## (a++)\*(a++)

int main()

{

int a = 5;

int b = 0;

b = (a++)\*(a++);

printf("b:%d\n", b);

return 0;

}

结果是２５

为什么不是30呢，因（a++）\*(a++)是一个乘法表达式，(a++)处理需要两条指令，第一条是获取a的值，++在第二条指令,而这个表达式在（a++）第一条指令就已经获取到值，所以++操作才会被置后处理．

实际上这段c代码被编译成汇编就成这样了：

　　movl $5, 28(%esp)

movl 28(%esp), %eax

imull 28(%esp), %eax

movl %eax, 24(%esp)

addl $1, 28(%esp)

addl $1, 28(%esp)

再成c翻译一下：

a=5:

b=a\*a;

a=a+1;

a=a+1;

或许其他编译器会有不同的解释．

## SPI

## IIC

## 堆和栈的区别

栈简介

栈由操作系统自动分配释放，用于存放函数的参数值、局部变量等，其操作方式类似于数据结构中的栈

int main()

{

int b;//栈

char s[] = "abc";//栈

char \*p2; //栈

}

其中函数中定义的局部变量按照先后定义的顺序依次压入栈中，也就是说相邻变量的地址之间不会存在其它变量。栈的内存地址生长方向与堆相反，由高到底，所以后定义的变量地址低于先定义的变量，比如上面代码中变量 s 的地址小于变量 b 的地址，p2 地址小于 s 的地址。栈中存储的数据的生命周期随着函数的执行完成而结束。

堆简介

堆由开发人员分配和释放， 若开发人员不释放，程序结束时由 OS 回收，分配方式类似于链表

（1）管理方式不同。栈由操作系统自动分配释放，无需我们手动控制；堆的申请和释放工作由程序员控制，容易产生内存泄漏；

（2）空间大小不同。每个进程拥有的栈大小要远远小于堆大小。理论上，进程可申请的堆大小为虚拟内存大小，进程栈的大小 64bits 的 Windows 默认 1MB，64bits 的 Linux 默认 10MB；

（3）生长方向不同。堆的生长方向向上，内存地址由低到高；栈的生长方向向下，内存地址由高到低。

（4）分配方式不同。堆都是动态分配的，没有静态分配的堆。栈有 2 种分配方式：静态分配和动态分配。静态分配是由操作系统完成的，比如局部变量的分配。动态分配由alloca()函数分配，但是栈的动态分配和堆是不同的，它的动态分配是由操作系统进行释放，无需我们手工实现。

（5）分配效率不同。栈由操作系统自动分配，会在硬件层级对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是由C/C++提供的库函数或运算符来完成申请与管理，实现机制较为复杂，频繁的内存申请容易产生内存碎片。显然，堆的效率比栈要低得多。

（6）存放内容不同。栈存放的内容，函数返回地址、相关参数、局部变量和寄存器内容等。当主函数调用另外一个函数的时候，要对当前函数执行断点进行保存，需要使用栈来实现，首先入栈的是主函数下一条语句的地址，即扩展指针寄存器的内容（EIP），然后是当前栈帧的底部地址，即扩展基址指针寄存器内容（EBP），再然后是被调函数的实参等，一般情况下是按照从右向左的顺序入栈，之后是被调函数的局部变量，注意静态变量是存放在数据段或者BSS段，是不入栈的。出栈的顺序正好相反，最终栈顶指向主函数下一条语句的地址，主程序又从该地址开始执行。堆，一般情况堆顶使用一个字节的空间来存放堆的大小，而堆中具体存放内容是由程序员来填充的。

函数调用压栈

## new和malloc

new和malloc都是动态分配内存的指令，它们的区别主要为以下五点：

1.申请内存所在位置不同

new从自由存储区上为对象动态分配内存空间

malloc从堆上动态分配内存

2.返回类型的安全性

new内存分配成功时，返回对象类型的指针，安全

Malloc成功时返回void\*，需要强转

3.分配内存失败时的结果不同

new分配内存失败时，会抛出bac\_alloc异常

malloc内存分配失败时返回NULL

4.是否需要指定内存大小

new申请内存分配时无需制定内存块的大小，编译器会根据类型自行计算

malloc需要显式地指出所需内存的大小

5.是否调用对象中的构造函数

new/delete会调用对象的构造/析构函数

malloc不会

6.属性方面

new是关键字，需要编译器支持；malloc是库函数，需要头文件支持。

7.处理数组方面

new有处理数组的new[]，使用new[]分配的内存必须使用delete[]进行释放。

int\* ptr=new int[100];//分配100个int的内存空间

1

malloc要想动态分配一个数组的内存，需要我们手动定义数组的大小。使用malloc分配内存必须使用free来释放内存。

int\* p=(int)malloc(sizeof(int)\*100);//分配可以放下100个int的内存空间

## 多态

函数名相同，函数形参也相同。调用时根据函数类型是[虚函数](https://so.csdn.net/so/search?q=%E8%99%9A%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/wufeifan_learner/article/details/_blank)还是普通成员函数决定调用哪一个。

编译时多态，运行时多态

如果在编译时能够确定执行多态方法称为编译时多态，否则称为运行时多态。

## 重载overload

函数名相同，但是函数参数不同。调用时根据参数的不同决定调用哪一个函数；

## 中断的上半部和下半部

上半部负责处理紧急的事情，不紧急的事情放在下半部。  
上半部对应的中断关闭，下半部对应的中断打开。

顶半部往往被设计成不可中断

底半部则相对来说并不是非常紧急的，而且相对比较耗时，不在硬件中断服务程序中执行。

中断触发、中断执行分开

中断执行时间尽可能短和中断处理需完成大量工作之间找到一个平衡点

如果一个任务对时间十分敏感，将其放在上半部。  
如果一个任务和硬件有关，将其放在上半部。  
如果一个任务要保证不被其他中断打断，将其放在上半部。

## 中断中使用睡眠会有什么影响

因为代码是运行在中断上下文中，并非进程上下文中，如果将中断进行睡眠的话，调度器无从得知下一个应该调度的进程，系统无法继续进行！

派生函数重写加不加override的区别

重载运算符

指针和引用的区别

声明的定义的区别

析构函数是否是虚函数

函数后面加const表示什么

vector和链表区别

unordered和mapde 底层原理

快排以及非递归写法

智能指针