- 内存布局
  - 栈
    - 从最高地址开始，向低地址分配
    - 保存函数调用维护信息（堆栈帧Stack Frame或活动记录Activate Record）
      - 函数的返回地址和参数
      - 非静态局部变量、编译器自动生成的其他临时变量(_cmd, self)
      - 上下文，包括函数调用后需要保持不变的寄存器
    - 函数调用流程
      - 1、保存寄存器ebp内容，然后将ebp指向栈顶，即mov ebp, esp(esp总是指向栈顶的，ebp用于保存函数进入时的栈顶，便于查找局部变量，函数参数等)
      - 2、在栈上开辟空间(sub esp, 0C0h)
      - 3、保存一些在函数调用结束后需要保持内容不变的寄存器内容
      - 4、加入调试信息
      - 5、函数返回，通过寄存器eax传递
      - 6、回复第三步保存的寄存器内容
      - 7、恢复esp和epb内容
      - 8、ret指令，返回。
    - 函数调用惯例
      - 函数调用方和被调用方对函数如何调用的统一理解
        - 参数的传递顺序和方式
          - 栈传递
          - 寄存器传递
        - 栈的维护方式
        - 名字修饰策略（指定调用惯例，例如：cdecl,stdcall,fastcall）
    - 函数返回值传递
      - 小于4字节，使用eax寄存器
      - 4-8字节，使用eax和edx联合返回
      - 大于8字节
        - 先在栈上开辟空间，称为temp对象
        - 把temp作为隐藏参数传入函数
        - 返回值拷贝值temp的内存空间上，并把temp的内存地址用eax传出
        - 把eax指向的temp内容拷贝给需要赋值的变量
      - 返回值内存空间过大时，要在栈上开辟空间并拷贝两次。
    - Hook
      - 标准进入指令序列
        - push ebp
        - move ebp, esp
      - 在标准进入指令序列前插入特殊内容
        - nop（占位）
        - mov edi, edi
      - 将特殊内容替换为jmp，实现hook
  - 堆
    - 容纳动态分配的内存区域(malloc, new)
    - 从低地址开始，向高地址增长。一般比栈大很多
    - 内存管理流程
      - 程序向操作系统申请一块适当大小的堆内存空间
      - 程序自己管理这块空间，不够时在向操作系统申请
      - 频繁向操作系统申请空间要调用内核，比较消耗性能，因此使用运行库统一向操作系统申请
    - 堆内存分配算法
      - 空闲链表Free LIst
        - 堆上空闲的块按照链表方式链接，请求空间时遍历整个列表，直到找到合适的大小进行拆分
        - 问题：一旦被破坏，整个堆无法使用。(容易被越界读写接触)
      - 位图（bitmap）
        - 整个堆分成大小相同的块，请求内存时总是分配整数个块的空间给用户
        - 头：已分配的块的第一块
主体（Body）：已分配区域的主体
          - 块只有三个状态：Free，Body，Head，因此两位即可表示一个块的状态(00, 01, 11)
        - 优点
          - 速度快，整个堆的空闲信息在一个数组内，访问时容易命中
          - 稳定性好
          - 易于管理
        - 缺点
          - 容易产生内存碎片
          - 堆很大，块很小时候，减少碎片，但是位图变大
      - 对象池
        - 可以用以上两种方式实现
        - 假定了每次分配空间都是同样大小
  - 可执行文件映像
    - 装载器将可执行文件的内存读取或映射在这里
  - 动态链接库映射区
    - 映射装载的动态链接库
  - 保留区
    - 不是单一内存区域，而是收到保护禁止访问的内存区域总称