C语言内存模型

1."栈溢出"

①定义:指程序在运行时，由于调用堆栈的深度超过了其所能容纳的极限，导致堆栈中的数据被错误地覆盖或丢失，进而引发程序异常或崩溃的现象

②原因:函数调用层次太深(函数递归调用层次过深，或者存在过多的嵌套函数调用)

动态申请空间未释放(申请的堆空间没有被及时释放，而系统又无法分配更多的堆空间给堆栈使用，

数组访问越界(程序中的数组访问超出了其定义的范围，可能会覆盖堆栈中的其他数据)

指针非法访问(指针保存了一个非法的地址)

③影响:程序异常或崩溃

④预防:优化函数调用(减少不必要的嵌套函数调用和递归调用，降低堆栈的使用深度)
及时释放动态空间(在C/C++等语言中，要及时释放动态申请的堆空间，避免占用过多内存资源)
加强数组和指针的访问控制确保数组访问在有效范围内，避免指针非法访问
使用栈保护技术

2.堆区和栈区的区别
①内存空间分配方式

栈区：由编译器自动分配和释放。当函数被调用时，栈区会为函数的局部变量和参数分配空间；当函数执行完毕后，这些空间会自动被释放。

堆区：由程序员手动分配和释放。程序员通过特定的函数来申请堆区内存，释放已分配的内存。

②内存空间大小和灵活性

栈区：栈区的大小通常由操作系统预先分配，相对较小且固定。由于栈区是连续的内存区域，因此其空间使用效率较高，但这也限制了栈区能够容纳的数据量和函数调用深度。

堆区：堆区的大小受限于计算机系统中有效的虚拟内存，因此其空间相对较大且灵活。堆区可以动态地分配内存，以适应不同大小和类型的数据结构需求。然而，这也增加了内存管理的复杂性，因为程序员需要手动管理内存的分配和释放。

③内存访问效率和缓存方式

栈区：栈区的内存访问效率通常较高，因为栈区是连续的内存区域，且由编译器自动管理。此外，栈区通常使用一级缓存来存储数据，进一步提高了数据访问速度。

堆区：堆区的内存访问效率相对较低，因为堆区是不连续的内存区域，且需要程序员手动管理。堆区的数据通常存储在二级缓存中，这增加了数据访问的延迟。

④数据结构和用途

栈区：栈区主要用于存储函数的局部变量、参数以及函数调用上下文等信息。栈区遵循“后进先出”（LIFO）的原则，即最后被压入栈中的数据会最先被弹出。

堆区：堆区主要用于存储动态分配的数据结构

3.只读和可读写区域

①只读区域:程序代码区，常量区，动态链接库

②可读写区域:全局数据区，堆区，栈区

4.malloc()、free()函数

①针对堆区操作

②malloc()函数
作用： `malloc()` 函数用于在运行时动态地分配一块指定大小的内存块，并返回这块内存的首地址。这块内存是未初始化的，其内容是未定义的。
头文件： `malloc()` 函数位于``头文件中。
语法： `void* malloc(size_t size);`
`size` ： 要分配的内存块的字节数。
返回值：成功时返回一个指向分配的内存块的指针；失败时返回一个空指针（ `NULL` ）

free()函数
作用： `free()` 函数用于释放之前通过 `malloc()` 或其他动态内存分配函数（如 `calloc()` 、 `realloc()` ）分配的内存块，以便系统可以重新利用这些内存空间。
头文件： `free()` 函数同样位于``头文件中。
语法： `void free(void* ptr);`
`ptr` ： 要释放的内存块的指针。这个指针必须是之前通过 `malloc()` 、 `calloc()` 或 `realloc()` 函数返回的指针。

5.程序内存管理意义

①促进资源有效利用:
使程序在运行时能够高效地利用这些资源，避免内存浪费，通过动态内存分配（如使用 `malloc()` 和 `free()` ），程序可以根据需要在运行时请求和释放内存，从而优化内存使用

②避免内存泄漏：长期的内存泄漏会耗尽系统的可用内存，导致程序性能下降，甚至系统崩溃

③提高程序稳定性:通过有效的内存管理，可以确保程序只访问已分配且有效的内存，从而提高程序的稳定性

④支持多任务处理：有效的内存管理可以确保每个任务都能获得所需的内存资源，同时避免任务之间的内存冲突

⑤优化程序性能：适当的内存分配和释放可以减少内存碎片，提高内存访问速度

⑥安全性:通过严格的内存管理，可以确保程序不会执行不安全的内存操作，从而提高程序的安全性

内存模型的运用

存储区域

constValue:常量区(只读区域段)，因为有const修饰且初始化了一个常量值，

constString:指针本身存储在数据段(全局/静态数据区)，字符串"Hello, World!"为常量，存储在只读数据段

globalVar:数据段(全局/静态数据区)，在整个程序运行期间始终存在且其值可以被访问和修改

staticVar:数据段(全局/静态数据区)，static修饰有静态存储期，存在于整个生命周期，但只作用域只在‘main’函数内部

localVar:栈区，局部变量在函数调用时会分配栈空间

ptr:作为指针本身存储在栈区，但其指向的内存区域是堆区

localVarMain:栈区，与localVar类似，同为局部变量且在‘main’函数调用时分配栈空间

浅谈Cache

1。①冯诺依曼体系结构是一种基本的计算机设计模型。这种架构的核心特征是程序和数据存储在同一存储空间（内存）中，并采用“存储程序”的原则，即计算机同时处理指令和数据。数据通过二进制方式进行表示和运算，程序按照固定的顺序执行

②现代计算机的组织结构通常被称为哈佛体系结构或混合型体系结构，它在冯诺依曼的基础上有所发展。

③哈佛体系结构将指令存储器（Instruction Memory, IM）和数据存储器（Data Memory, DM）分开，提高了读取速度。而混合型结构则结合了两者的特点，有时会有一个统一的地址空间，但在某些硬件层面仍保留了部分分离。

2.主存储器，通常指RAM（随机存取内存），工作原理是通过地址线寻址，一次只能访问一个存储单元，读写操作几乎瞬时完成，但访问速度受限于存储介质的物理特性。

3.①Cache的局部性原理是指，程序在运行过程中倾向于频繁地访问最近使用的数据和指令。

②这包括时间局部性和空间局部性。时间局部性指连续执行的指令往往来自相近的时间点；空间局部性指程序在短时间内可能会多次访问同一块内存区域的数据。为了利用这个原理，Cache设计成高速、

小容量但高访问速度的存储，优先缓存最近使用的内容。

4.Cache的应用之所以能提升系统性能，是因为它可以减少对主存储器的依赖。由于Cache的快速响应，大部分常用数据可以在极短的时间内从Cache中获取，从而大大减少了数据传输延迟，提高了系统的整体效率。当程序需要的数据不在Cache中，才去慢速的主存查找，这就是所谓的“层次式存储”策略。

代码优化
见附件