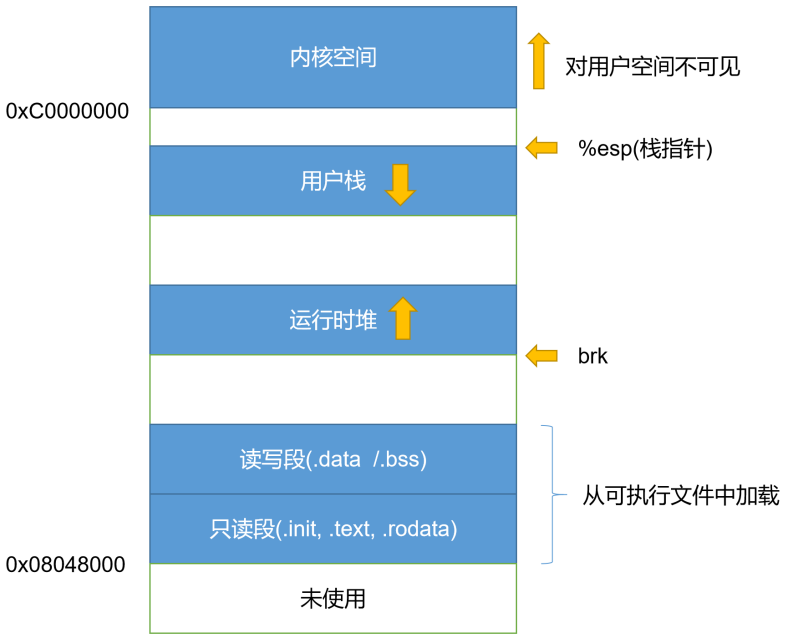
程序内存分配方式

程序内存布局

现在的应用程序都运行在一个虚拟内存空间里，以32位系统为例，其寻址空间为4G，大部分的操作系统都将4G内存空间的一部分挪给内核调用，应用程序无法直接访问这一段内存，这一部分内核地址成为内核态空间，Linux默认将高地址的1G空间分配给内核，用户使用剩下的3G空间成为用户态空间，用户态空间一般有如下默认区域：

1. 栈区(stack)：由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。
2. 堆区(heap)：一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构 中的堆是两回事，分配方式倒是类似于链表。
3. 全局/静态区(static) ：全局变量和静态变量的存储是放在一块的，在程序编译时分配。
4. 文字常量区：存放常量字符串。
5. 程序代码区：存放函数体（类的成员函数、全局函数）的二进制代码

虚拟内存空间示意图如下：



虚拟内存空间示意图

当执行一个非内置的可执行程序时，加载器会将可执行目标文件中的代码和数据从磁盘拷贝到内存中。每个Linux程序都会有一个运行时存储器映像，如上图所示。在可执行文件中段头部表的指导下，加载器将可执行文件的相关内容拷贝到代码和数据段。

- 在32位系统下，代码段总是从地址0x08048000处开始。
- 数据段是在接下来的下一个4KB对齐的地址处。
- 运行时堆(heap)在读写段之后接下来的第一个4KB对齐的地址，然后向上增长。
- 用户栈(stack)从最大的合法用户地址开始，向下增长。
- 从3G地址往上的部分是由内核使用的。

程序验证

请看下面这段程序，判断其中的变量位于哪个区域

```
int a = 0;
char *p1;
int main(){
    int b;
    char s[] = "123456";
    char *p2;
    char *p3 = "123456";
    static int c = 0;
    p1 = new char[10];
    p2 = new char[5];
    strcpy(p1, "123456");
}
```

栈与堆的比较

申请后系统的响应

- 栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。
- 堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。另外，对于大多数系统，首地址处会记录这块内存空间中本次分配的大小，这样，代码中的delete语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

申请效率的比较：

- 栈由系统自动分配，速度较快。但程序员无法控制。
- 堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便。

申请大小的限制

- 栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。
- 堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。。由此可见，堆获得的空间比堆的大小受限于计算机系统中有效的虚拟内存较灵活，也比较大。

堆和栈中的存储内容

- 栈： 在函数调用时，第一个进栈的是主函数的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。
- 堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容由程序员安排。

栈与堆的区别

栈与堆的区别在一下六个方面有所不同：

1. 管理方式不同。对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生memory leak.
2. 空间大小不同。一般来讲在32位系统下，内存可以达到4G的空间，从这个角度来看堆内存几乎是没有什么限制的。但是对于栈来讲，一般都是有一定的空间大小的，例如，在VS下，默认的栈空间大小是1M
3. 分配方式。内存有2种分配方式：静态分配和动态分配。堆都是动态分配的，没有静态分配的堆。静态分配是编译器完成的，比如局部变量的分配。动态分配由malloc, calloc函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。
4. 生长方向。对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长。

5. 碎片问题。对于堆来讲，频繁的new/delete势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个问题，因为栈是先进后出的，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出，在他弹出之前，在它上面的后进的栈内容已经被弹出。
6. 生长方向。对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长。