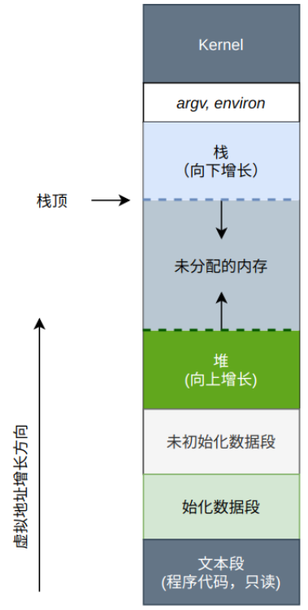
进程内存空间

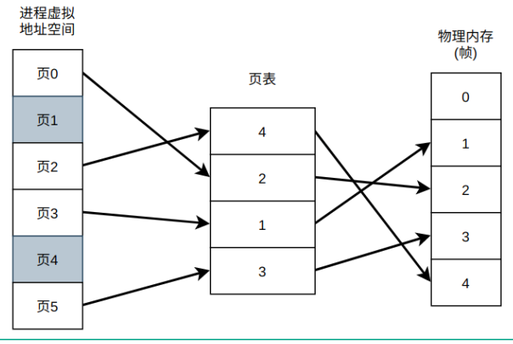
进程内存布局

- 文本段 — 用于保存进程运行的程序机器语言指令，CPU中的程序计数寄存器则指向该段内存位置
- 初始化数据段 — 包含显式初始化的全局变量和静态变量，当可执行文件载入内存时，可执行文件直接读取这些变量的值
- 未初始化数据段 — 包含了未进行显式初始化的全局变量和静态变量，出于历史原因，此段常被称之为BSS段
- 栈(stack) — 是一个动态增长和收缩的端，有栈帧(frame)组成。内核会为每个当前调用的函数分配一个栈帧，栈帧中保存了函数的局部变量、实参和返回值
  - 通常一个函数调用完毕后内核将该函数栈帧从栈中弹出，以释放内存
- 堆(heap) — 能够在运行时动态进行内存分配的一块区域，当用户未主动释放于堆中申请的内存时，数据将会一值存在



虚拟内存管理

- 上图中的进程内存结构并不完全处于内存之中，可能有一部分数据处于磁盘之中
- 现代操作系统为了使得系统能够最大化的利用硬件性能，如内存、CPU，并尽可能多的运行程序，抽象出了虚拟内存
- 虚拟内存空间通常会大于实际的物理可用内存，为了实现这一点，操作系统将虚拟内存空间以页的形式进行划分，大小通常为4096字节。并建立虚拟内存页与物理内存帧的映射，该映射通常称之为页表



图中灰色页为未建立页表项的虚拟内存页，当程序访问该段虚拟内存时，会引发内核的缺页异常，内核将建立该虚拟页与物理内存的映射表项

栈和栈帧

每次的函数调用时，将会在栈上重新分配一帧，每当函数返回时，再从栈上将此帧移去 — 通常而言，当栈帧被移除时，栈会收缩，栈的大小将会减少

但是在大多数Linux实现上，当函数返回时，栈的大小并不会减少，而是在后续函数调用中进行覆盖

```c
int *func() {
    int n = 10;
    return &n;  // Wrong!
}
```