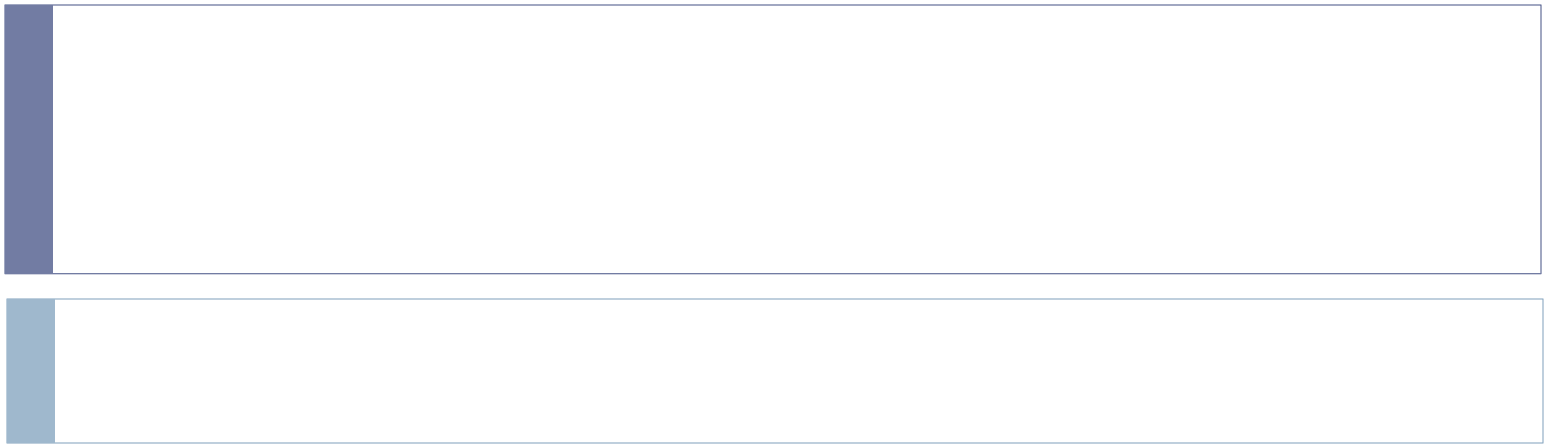
Chapter 8 运行时存储空间组织



Outlines

- 符号表的组织与作用
- 整理与查找
- 名字的作用范围
- 符号表的内容
- 存储组织与管理
- 目标程序运行时的活动
- 运行时存储器的划分
  - 存储分配策略

Symbol table

- 源程序中各种名字的属性和特征等有关信息
  - 编译器需要不断汇集和反复查证
  - 通常记录在一张或几张符号表中
  - 用于语义检查、中间代码和目标代码生成等阶段
- 提高编译器工作效率的重要一环
  - 合理地组织符号表
  - 减少符号表本身占据的存储空间
  - 提高符号表的访问效率

Symbol table

符号名表	常量名、变量名、数组名、过程名、 性质、引用、定义
常数表	各种类型常数的值
标号表	标号的定义和引用情况
入口名表	过程的入口名和入口位置
过程引用表	外部过程的名字、引用位置

循环表

等价名表

公用链表

格式表

中间代码表

Symbol table

- 符号表的作用
  - 一致性检查和作用域分析
  - 辅助代码生成
- 符号表的每一项(入口)包含两大栏
  - 名字栏，也称主栏，关键字栏
  - 信息栏，记录相应的不同属性

符号名表

NAME	INFORMATION
M	形参，整型值参数
N	形参，浮点型值参数
K	整型变量

Symbol table

- 对符号表的操作
  - 查找名字
  - 填入名字
  - 访问给定名字的某些信息
  - 填写或修改给定名字的某些信息
  - 删除一个或一组无用的项

Symbol table

- 对符号表进行操作的时机：
  - 定义性出现
  - 使用性出现
- 按名字的不同种属建立多张符号表，如
  - 常数表
  - 变量名表
  - 过程名表
  - …
- 符号表的组织方式
  - 安排各项各栏的存储单元为固定长度
  - 用间接方式安排各栏存储单元

Symbol table

- 设符号表可容纳N项，每项需用K个字
  - 把每一项置于连续K个存储单元中
    - 构成一张K*N的表
  - 把整个符号表分成m个子表，如$T_1, T_2, \ldots T_m$
    - 每个子表含有N项

Search

- 按关键字出现的顺序填写各项
- 填表快，查找慢
- 结构简单，节省空间
- 效率低
  - 查找时间复杂度：*O(n)*
- 改进：自适应线性表
  - “最新最近”访问原则

Search

- 二分查找
  - 表格中的项按名字的“大小”顺序整理排列。
  - 填表慢,查找快
  - 查找时间复杂度：$O(log_2 n)$
- 改进：二叉树查找

Search

- 原理：HASH技术
  - 哈希函数H(SYM)
    - 0~N-1（N是符号表的项数）
    - 计算简单高效
    - 函数值分布均匀
- 填表快，查找快

Symbol table

- 在许多程序语言中
  - 名字都有一个确定的作用范围
  - 作用域分析
- 名字的作用域规则
  - 规定一个名字在什么样的范围内应该表示什么意义的原则
  - 常用方法：最近嵌套作用域规则
    - 对每个过程指定一个唯一的编号
    - 在符号表中用一个二元组<名字,过程编号>表示名字
    - 在查找每个名字时,先查对过程编号,确定所属的表区段落,然后从该段落中查对标识符

Content

- 符号表的信息栏中登记了每个名字的有关性质
  - 类型：整型、实型或布尔型等
  - 种属：简单变量、数组、过程等
  - 大小：长度，即所需的存储单元字数
  - 相对数：指分配给该名字的存储单元的相对地址

Run-time environment

- 编译程序最终目的是
  - 将源程序翻译成等价的目标程序

- 除了对源程序进行词法、语法和语义分析外, 生成目标代码前
  - 需要把程序及其运行活动联系起来
  - 弄清将来在代码运行时刻
    - 源代码中各种变量、常量等如何存放
    - 如何去访问它们

Run-time environment

- 在程序的执行过程中
  - 数据存取通过与之对应的存储单元进行
- 在程序语言中
  - 程序中使用的存储单元都由标识符来表示
- 存储组织与管理
  - 活动记录的建立与管理
  - 存储器的组织
  - 存储分配策略
  - 非局部名称的访问等

Run-time environment

- 假定程序由若干个过程（procedure）组成
  - 过程定义
    - 关联一个标识符（过程名）和一段语句（过程体）
  - 一个过程的活动指的是该过程的一次执行
  - 过程P一个活动的生存期，指的是从执行该过程体第一步操作到最后一步操作之间的操作序，包括执行P时调用其它过程花费的时间
    - 生存期：在程序执行过程中若干步骤的一个顺序序列
  - 过程可以是递归的

Run-time environment

- 过程（函数）
  - 是模块程序设计的主要手段
  - 也是节省程序代码和扩充语言能力的主要途径
- 只要过程有定义，就可以在别的地址调用它
- 调用与被调用（过程）两者之间的信息往来
  - 或者通过全局量
  - 或者经由参数传递

Parameter passing

- 传地址
  - 把实参的地址传递给相应的形参
- 得结果
  - 传地址的一种变形
  - 每个形参对应两个形式单元
    - 第一个形式单元存放实参的地址
    - 第二个单元存放实参的值
- 传值
  - 将实参的值传递给相应的形参
- 传名
  - 特殊方式（ALGOL60）

Run-time environment

- 为了组织存储空间，编译程序中必须考虑
  - 过程是否允许递归?
  - 当控制从一个过程的活动返回时，对局部名称的值如何处理?
  - 过程是否允许引用非局部名称?
  - 过程调用时如何传递参数；过程是否可以做为参数被传递和做为结果被返回?
  - 存储空间可否在程序控制下进行动态分配?
  - 存储空间是否必须显式地释放?

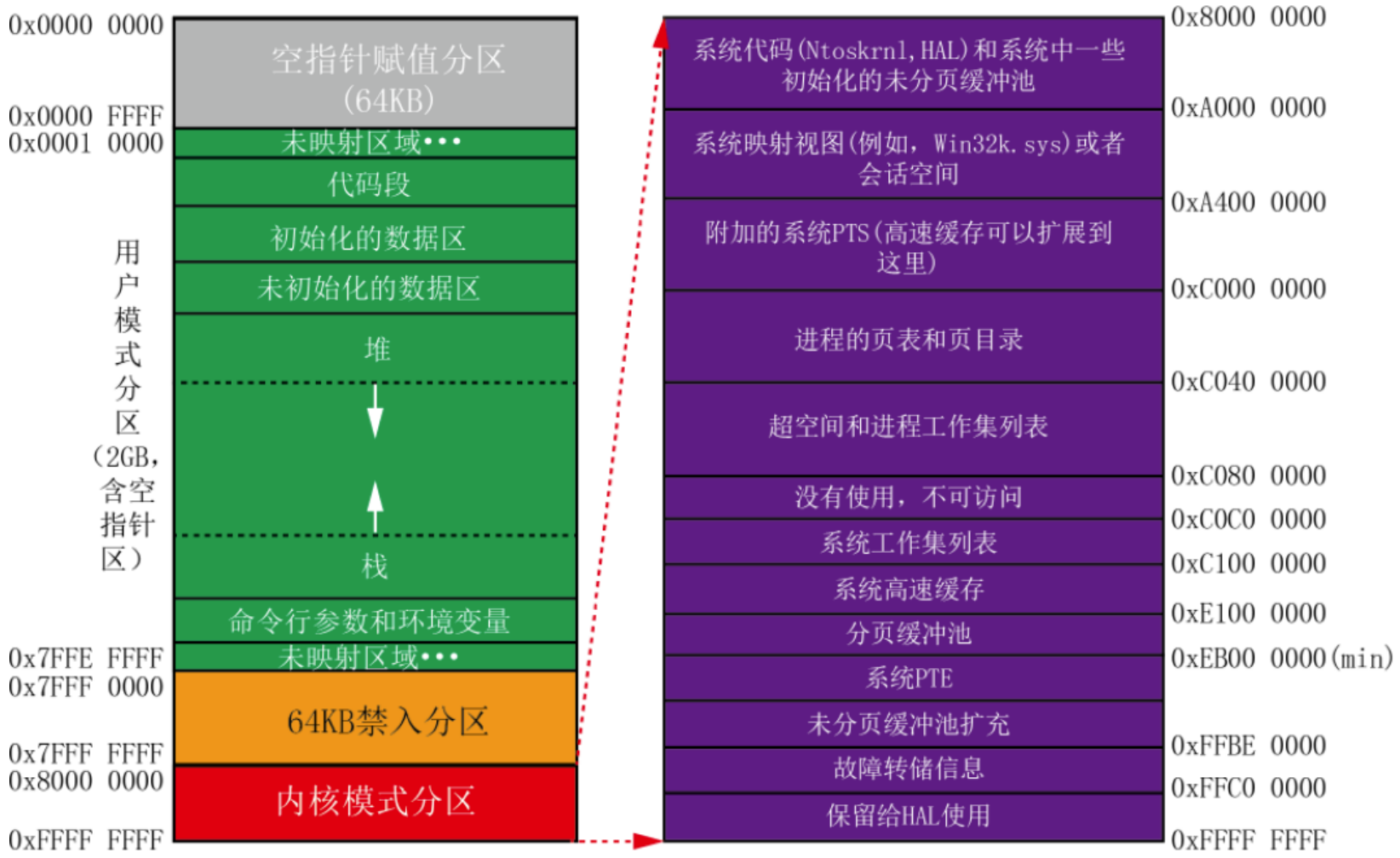
windows内存管理

表13-1　进程的地址空间如何分区

分　区	32位Windows 2000（x86和Alpha处理器）	32位Windows 2000(x86 w/3 GB用户方式）	64位Windows 2000（Alpha和IA-64处理器）	Windows 98
NULL指针分配的分区	0x00000000 0x0000FFFF	0x00000000 0x0000FFFF	0x00000000 00000000 0x00000000 0000FFFF	0X00000000 0x00000FFF
DOS/16位Windows应用程序兼容分区	无	无	无	0x000001000 0x003FFFFF
用户方式	0x00010000 0x7FFEFFFF	0x00010000 0xBFFEFFFFF	0x00000000 00010000 0x000003FF FFFEFFFF	0x00400000 0x7FFFFFFF
64-KB禁止进入	0x7FFF0000 0x7FFFFFFF	0xBFFF0000 0xBFFFFFFF	0x000003FFFFFF0000 0x000003FFFFFFFFFF	无
共享内存映射文件（MMF) 内核方式	无 0x800000000 0xFFFFFFFF	无 0xC0000000 0xFFFFFFFF	无 0x00000400 00000000 0xFFFFFFFFF FFFFFFF	0x80000000 0xBFFFFFFF 0xC0000000 0xFFFFFFFF

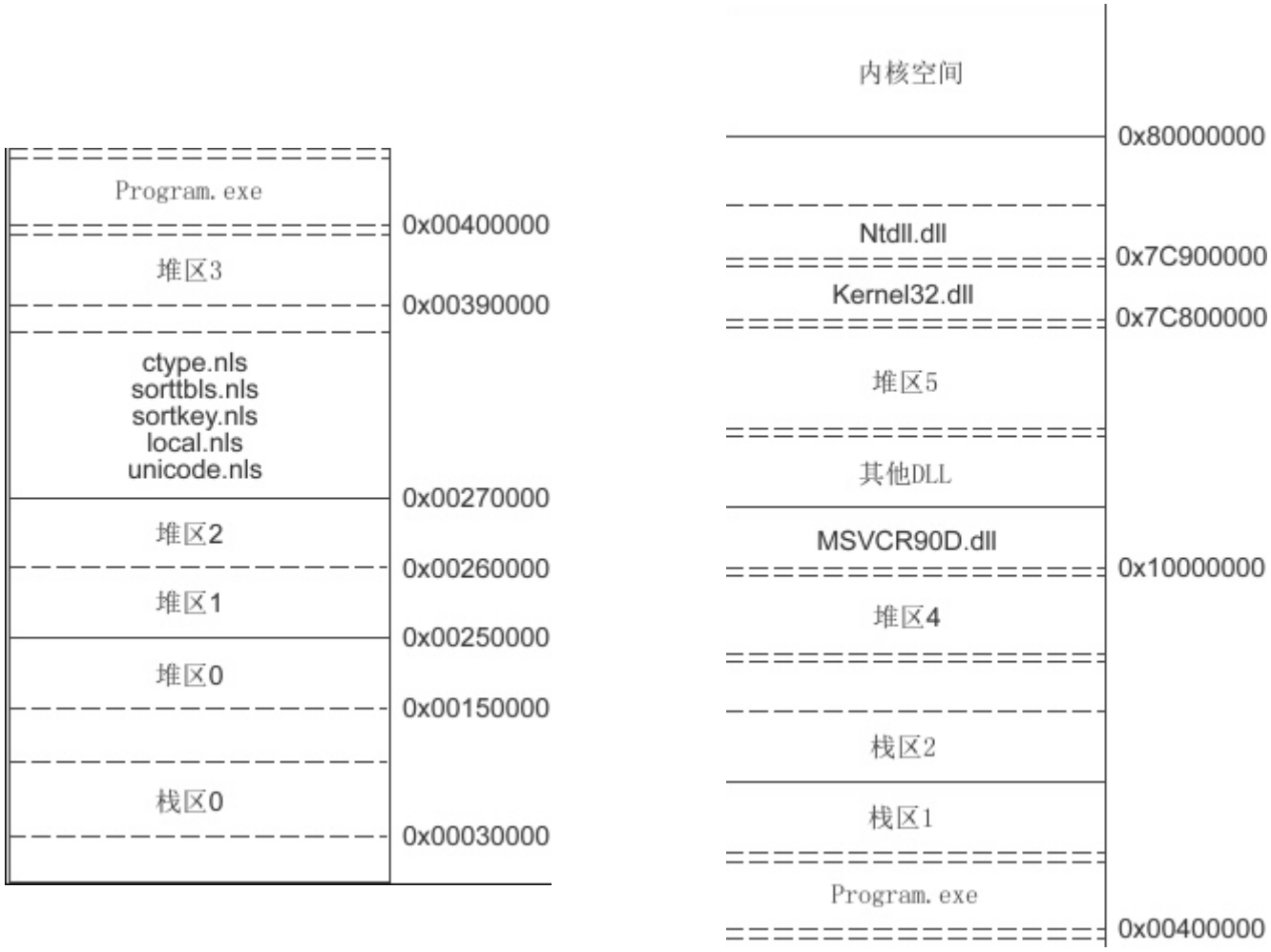
来源：windows核心编程

内存布局

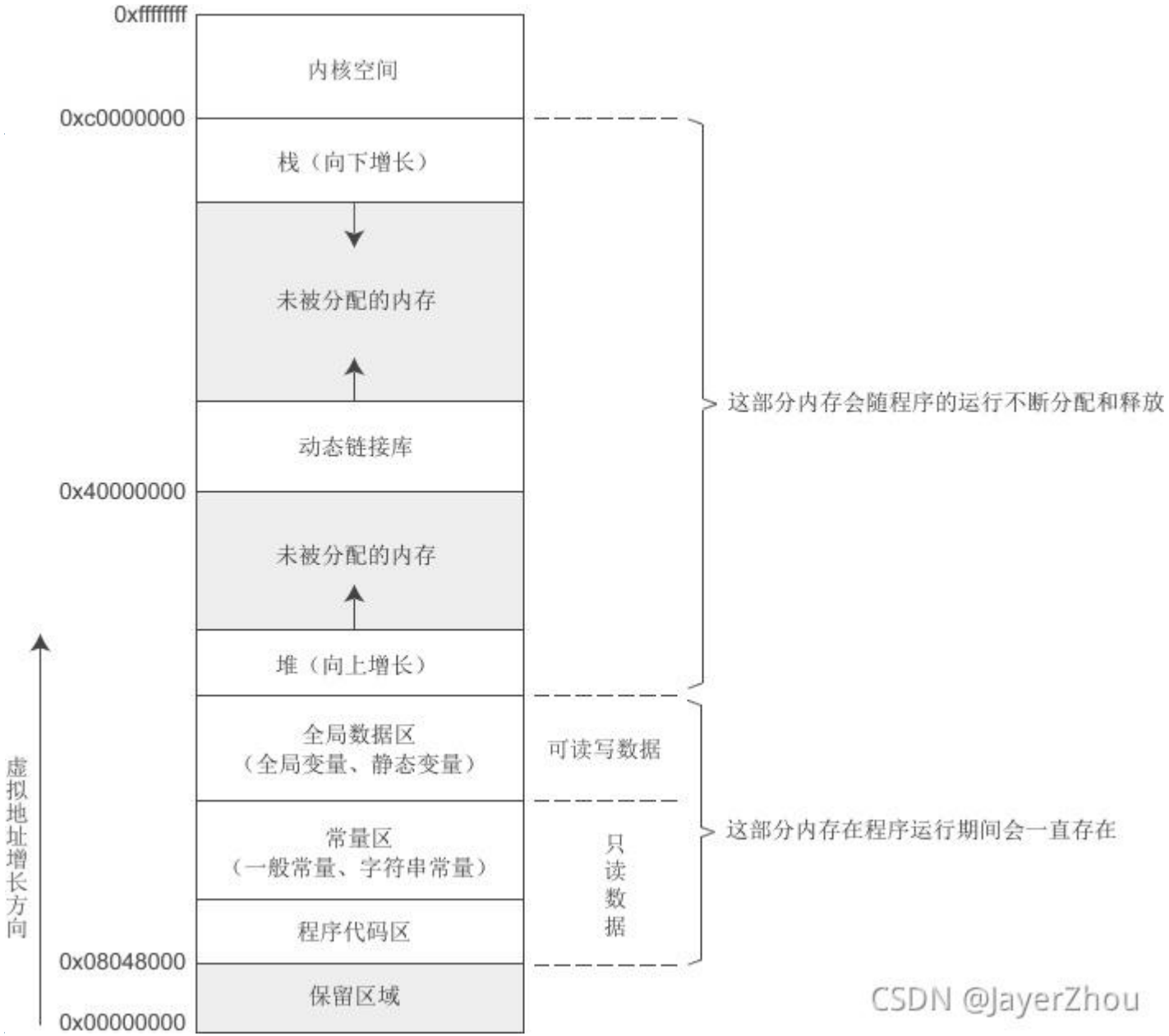


x86系统32位Windows内存空间布局

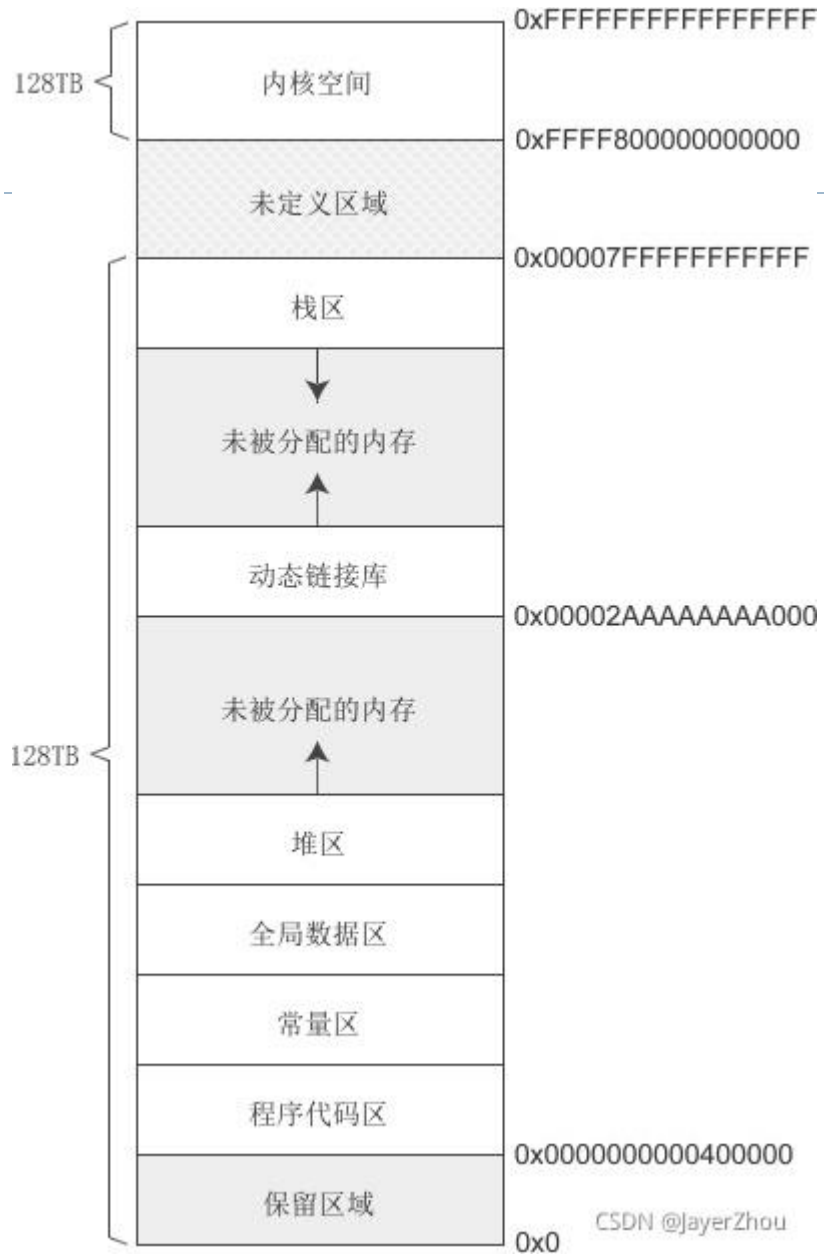
内存布局


Program.exe
0x00400000
堆区3
0x00390000
ctype.nls
sorttbls.nls
sortkey.nls
local.nls
unicode.nls
0x00270000
堆区2
0x00260000
堆区1
0x00250000
堆区0
0x00150000
栈区0
0x00030000
内核空间
0x80000000
Ntdll.dll
0x7C900000
Kernel32.dll
0x7C800000
堆区5
其他DLL
MSVCR90D.dll
0x10000000
堆区4
栈区2
栈区1
Program.exe
0x00400000

内存布局



内存布局


0xFFFFFFFFFFFFFFFF
128TB
内核空间
0xFFFF800000000000
未定义区域
0x00007FFFFFFFFFFF
栈区
未被分配的内存
动态链接库
0x00002AAAAAAAA000
未被分配的内存
128TB
堆区
全局数据区
常量区
程序代码区
0x0000000000400000
保留区域
0x0
CSDN @JayerZhou

参考资料

- https://blog.csdn.net/weixin_28843191/article/details/117051088
- https://blog.csdn.net/qq_37655329/article/details/121495298
- https://blog.51cto.com/u_15072778/3993250

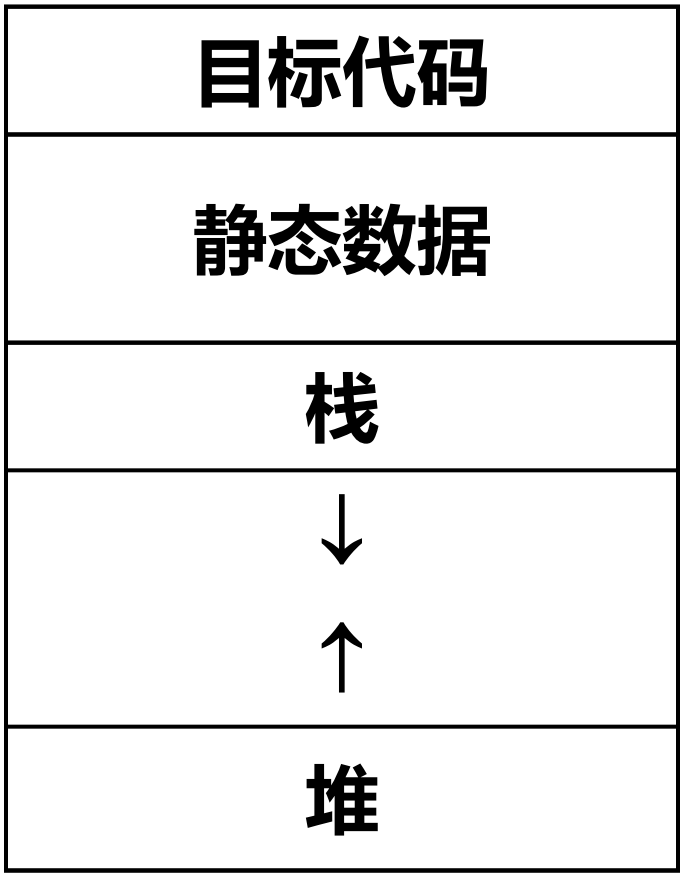
Run-time environment

- 目标程序运行所需的存储空间
  - 存放目标代码的空间
  - 存放数据对象的空间
  - 存放控制栈的空间
    - 程序运行的控制或连接数据所需单元
  - 堆（heap）
    - 用来存放动态数据

栈或堆的大小

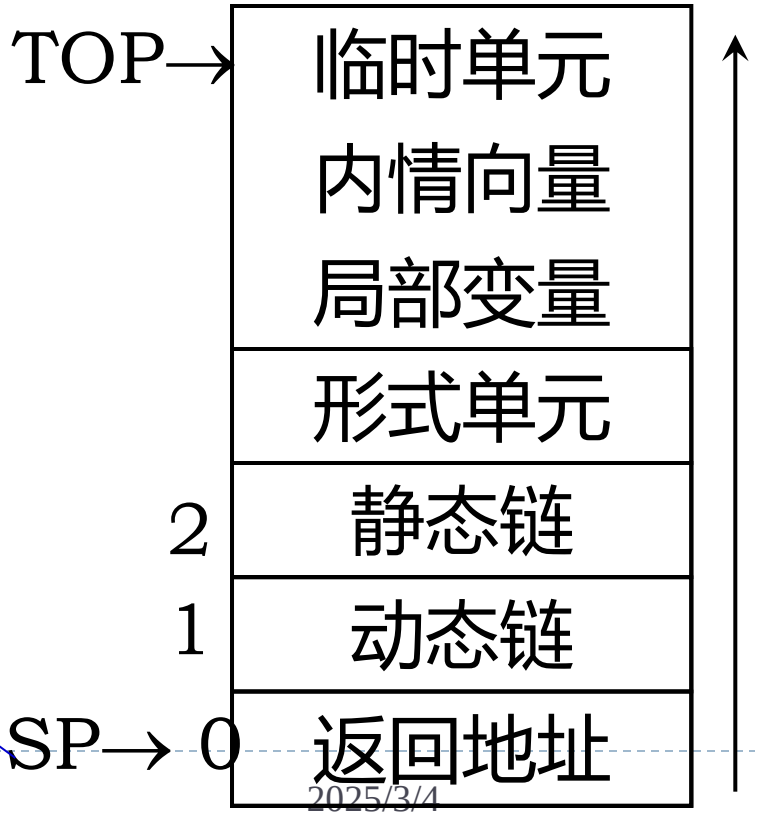
都随程序的运行而改变

应使它们的增长方向相对



Run-time environment

- 一个连续的存储块
  - 用来管理过程在一次执行中所需要的信息
  - 每进入一个过程，就有一个活动记录累筑于栈顶
- 活动记录结构
  - 连接数据
    - 返回地址
    - 动态链
    - 静态链
  - 形式单元
  - 局部变量
  - 内情向量
  - 临时工作单元等



指向现行过程的活动记录在栈里的起始位置

Run-time environment

- 静态分配策略
  - 在编译时对所有数据对象分配固定的存储单元
  - 且在运行时始终保持不变
- 栈式动态分配策略
  - 在运行时把存储器作为一个栈进行管理
    - 每当调用一个过程，所需存储空间就动态地分配于栈顶
    - 一旦退出，所占空间就予以释放
- 堆式动态分配策略
  - 在运行时把存储器组织成堆结构
    - 凡申请者从堆中分给一块
    - 凡释放者退回给堆

Exercise

- 例1：下面的程序执行时，输出的a分别是什么？若参数的传递办法分别为(1)传名；(2)传地址；(3)得结果；4)传值。

```
program main (input,output);
   procedure p(x,y,z);
   begin
         y:=y+1;
         z:=z+x;
   end;
begin
   a:=2;
   b:=3;
   p(a+b,a,a);
print a
end.
```

- 解：
  - (1) 参数的传递办法为“传名”时，a 为 9。
  - (2) 参数的传递办法为“传地址”，a 为 8。
  - (3) 参数的传递办法为“得结果”，a 为 7。
  - (4) 参数的传递办法为“传值”，a 为 2。

Exercise

- 例2：过程参数的传递方式有几种？简述“传地址”和“传值”的实现原理。
- 解：参数的传递方式有下述几种：传值，传地址，传名，得结果
  - “传值”方式，这是最简单的参数传递方法。即将实参计算出它的值，然后把它传给被调过程。具体来讲是这样的：
    - 1.形式参数当作过程的局部变量处理，即在被调过程的活动记录中开辟了形参的存储空间，这些存储位置即是我们所说的实参或形式单元。
    - 2.调用过程计算实参的值，并将它们的右值（r-value）放在为形式单元开辟的空间中。
    - 3.被调用过程执行时，就像使用局部变量一样使用这些形式单元。
  - “传地址”方式，也称作传地址，或引用调用。调用过程传给被调过程的是指针，指向实参存储位置的指针。
    - 1.如实参是一个名字或是具有左值的表达式，则左值本身传递过去。
    - 2.如实参是一个表达式，比方a+b或2，而没有左值，则表达式先求值，并存入某一位置，然后该位置的地址传递过去。
    - 3.被调过程中对形式参数的任何引用和赋值都通过传递到被调过程的指针被处理成间接访问。