第9章 运行时存储组织

- 主要内容
  - 运行时存储组织的任务
  - 运行时刻的内存划分(Partition)
  - 活动记录与过程调用
  - 存储分配策略
    - 静态分配策略
    - 栈式(Stack)动态存储分配策略
    - 堆式(Heap)动态存储分配策略

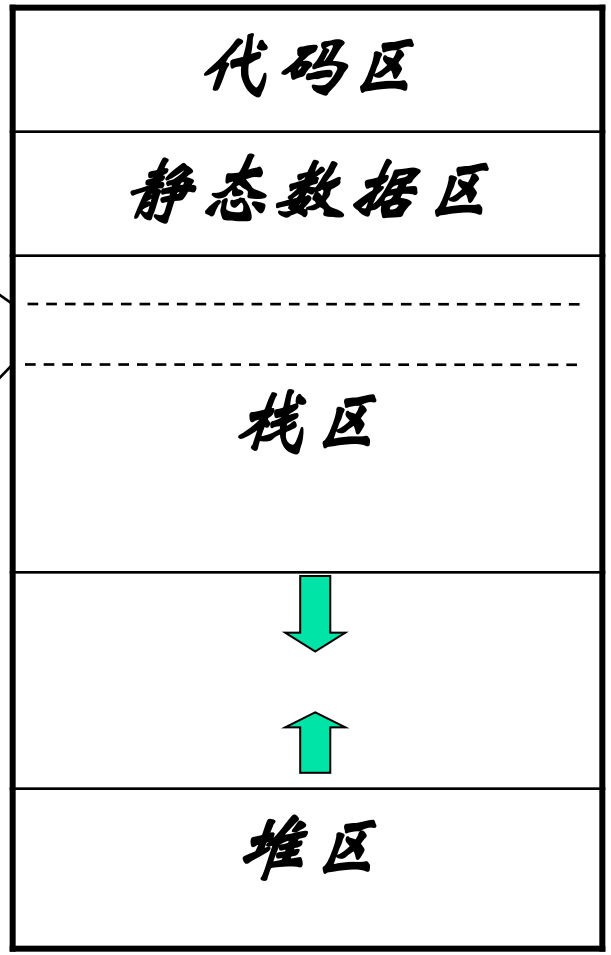
9.1.1运行时刻存储组织概述

- 运行时刻环境
  - 编译器支持各种抽象概念：名字、作用域、数据类型、运算符、过程、参数…
  - 编译器需要和操作系统等系统软件协作，在目标机上实现上述概念
  - 运行时刻环境：目标程序运行的环境，包括命名对象的分配和安排存储位置、确定目标程序访问变量时使用的机制、过程间的连接、参数传递、与操作系统接口、输入输出接口等。

9.1.2运行时刻的内存划分

局部数据区中的一个栈单元

——活动记录(静态/动态分配)



9.1.3 运行时的存储分配策略

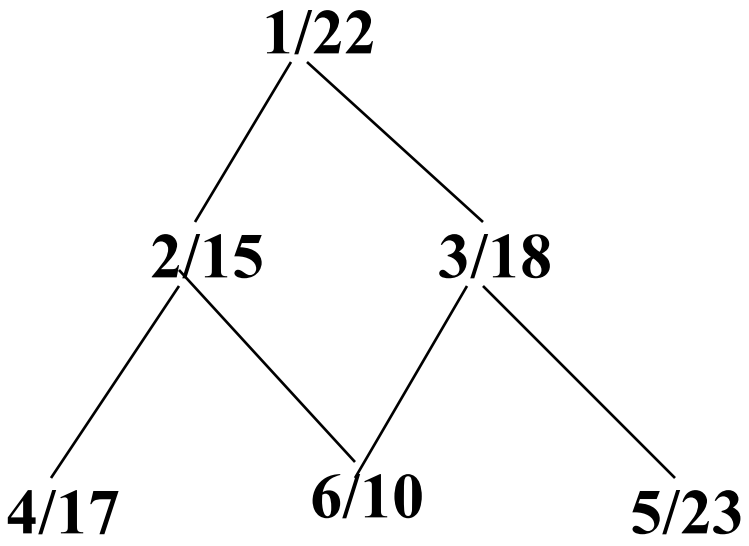
- static：对应编译时刻
- dynamic：对应运行时刻
- 存储分配的策略有静态分配与动态分配两类.
  - 静态分配：编译时已经确定好的量，合于无动态申请内存,无可变体积数组,无递归调用的程序.如FORTRAN,BASIC等.
  - 动态存储分配适用面广是目前最常用的分配方案
  - 动态分配又有栈式分配与堆式分配两种.

1.静态存储分配

- 特点
  - 编译时刻确定存储位置
  - 访问效率高
- 主要用途
  - 子程序的目标代码段
  - 全局数据目标（全局变量）
- ？？ 用什么样的算法实现静态存储分配

静态存储分配策略介绍

- 顺序分配算法
  - 按照程序段出现的先后顺序逐段分配



程序段之间的调用关系

程序段号/所需数据空间

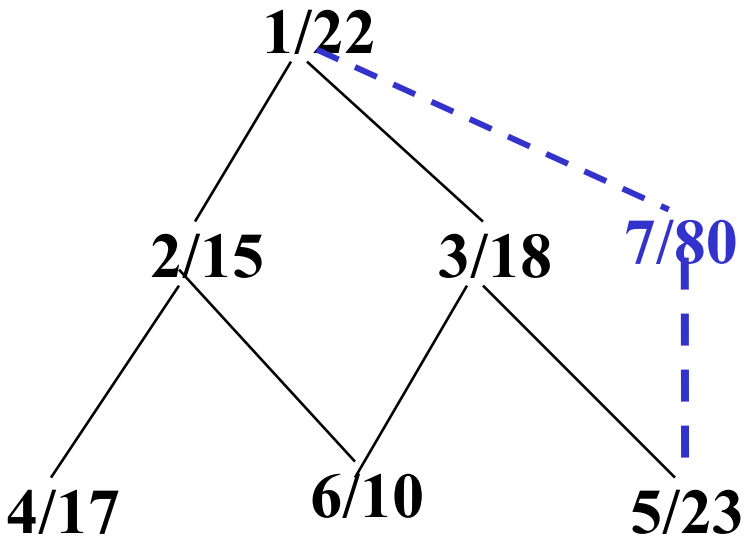
程序段	区域
1	0~21
2	22~36
3	37~54
4	55~71
5	72~94
6	95~104

共需要105个存储单元

能用更少的空间么？

静态存储分配策略介绍

允许程序段之间的覆盖（覆盖可能性分析）



程序段	区域
6	0~9
5	0~22
4	0~16
3	23~40
2	17~31
1	41~62

原始总存储需求=105个存储单元

共需要63个存储单元

思考：如何设计分配算法?

静态存储分配无法克服的问题

- 动态数组问题
- 递归调用问题
  - 栈式存储分配
- 动态申请存储空间问题
  - 堆式存储分配
- 被调用者的生存期超过调用者/局部数据需要保留（save）
  - 堆式存储分配

栈（Stack）式存储分配

- 用于有效实现可动态嵌套的程序结构
  - 如实现过程/函数，块层次结构
- 可以实现递归过程/函数
  - 比较：静态分配不宜实现递归过程/函数
- 运行栈中的数据单元是活动记录（后面介绍）

栈（Stack）式存储分配

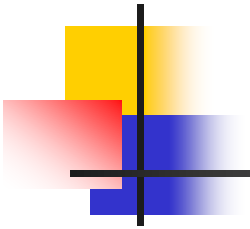
- 过程调用和返回由一个运行时刻的栈管理，这个栈称为运行栈。
- 运行栈中的每个记录用于管理每个活动，称为一个活动记录或栈帧
- 特点
  - 嵌套调用次序
  - 先进后出
  - 生存期限于本次调用
  - 自动释放

堆（Heap）式存储分配

- 用于动态数据结构
  - 存储空间的动态分配和释放
- 实现方法：
  - 将内存空间分为若干块，根据用户要求分配
  - 无法满足时，调用无用单元收集程序将被释放的块收集起来重新分配
- 存储管理器（存储空间分配和回收）

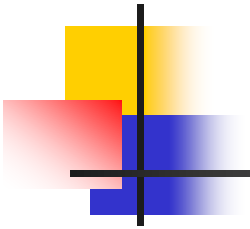
堆（Heap）式存储分配

- 从堆空间为数据对象分配/释放存储
  - 灵活 数据对象的存储分配和释放不限时间和次序
- 显式的分配或释放（*explicit allocation / dealocation*）
  - 程序员负责应用程序的（堆）存储空间管理（借助于编译器与（或）运行时系统所提供的默认存储管理机制）
- 隐式的分配或释放（*implicit allocation / dealocation*）
  - （堆）存储空间的分配或释放不需要程序员负责，由编译器与（或）运行时系统自动完成



堆（Heap）式存储分配

- 某些语言有显式的堆空间分配和释放命令
  - 如：Pascal 中的 *new*, *deposit*
    C++ 中的 *new*, *delete*
  - 比较：C 语言没有堆空间管理机制，*malloc()* 和 *free()* 是标准库中的函数，可以由 *library vendor* 提供
- 某些语言支持隐式的堆空间释放
  - 采用**垃圾回收**（*garbage collection*）机制
  - 如：Java 程序员不需要考虑对象的析构



堆（Heap）式存储分配

- 不释放堆空间的方法
  - 只分配空间，不释放空间，空间耗尽时停止
  - 适合于堆数据对象多数为一旦分配，永久使用的情形
  - 在虚存很大及无用数据对象不致带来很大零乱的情形也可采用

堆式存储分配

一 堆空间的管理

- **分配算法** 面对多个可用的存储块，选择哪一个

  如：**最佳适应算法**（选择浪费最少的存储块）

  最先适应算法（选择最先找到的足够大的存储块）

  循环最先适应算法（起始点不同的最先适应算法）

- **碎片整理算法** 压缩合并小的存储块，使其更可用

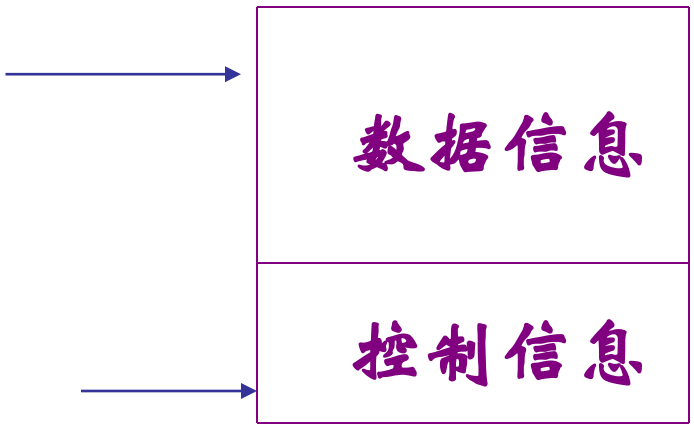
（可以分专门的话题讨论，超出本课程范围）
（部分内容可参考数据结构和操作系统课程）

9.2 活动记录

一 过程活动记录

- 函数/过程调用或返回时，在运行栈上创建或从运行栈上消去的**栈帧**（*frame*），包含**局部变量**，**函数实参**，**临时值**（用于表达式计算的中间单元）等数据信息以及必要的**控制信息**

某个数据对象的地址＝
活动记录起始地址
＋ 偏移地址（*offset*）



活动记录起始地址

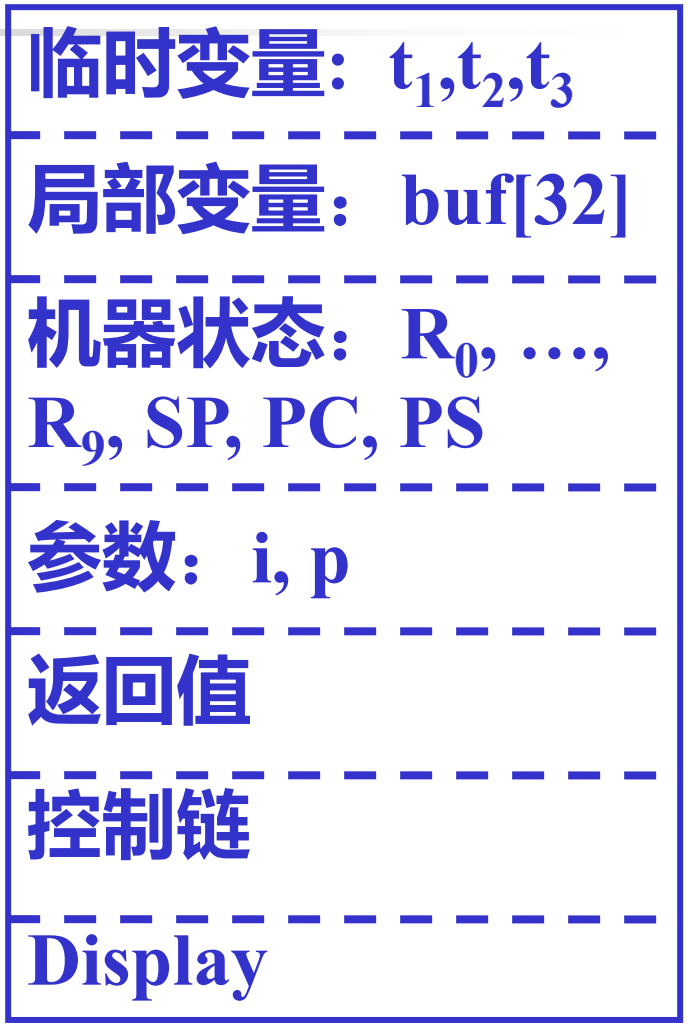
活动记录中过程所用信息

临时变量
局部变量
机器状态
实在参数
返回值
控制链
访问链

- 用于表达式的计算
- 局部数据
- 寄存器、程序计数器（返回地址）
- 保存实在参数的值或地址
- 存放返回值
- 保存调用者活动记录地址等(SP)
- 用于存取嵌套外层过程中的非局部数据

例子——函数的活动记录

```
int sub( i, p )
int  i;
char *p;
{
  char  buf[32];
  buf[i] = *(p + i);
  return i + 1;
}
```

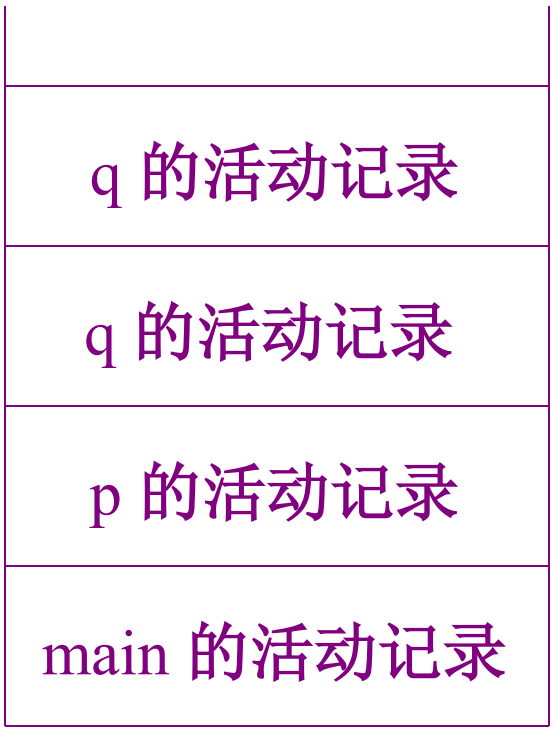


活动记录的管理

– 过程活动记录的栈式分配举例

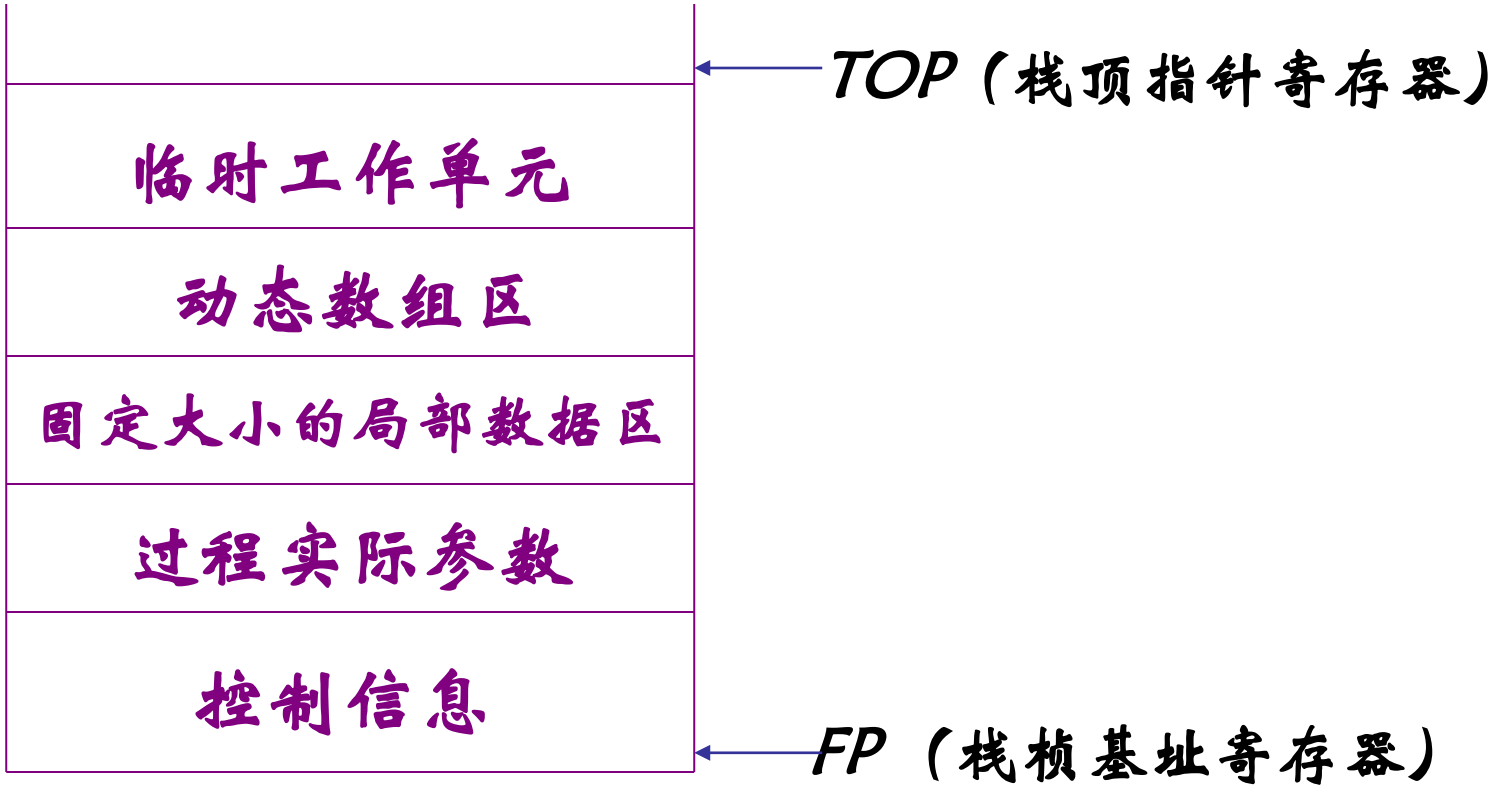
```
void p( )  {
   …
   q( );
}

void q( )  {
   …
   q( );
}

int main  {
   p( );
}
```

函数 q 被第二次激活时运行栈上活动记录分配情况



活动记录

– 典型的过程活动记录结构

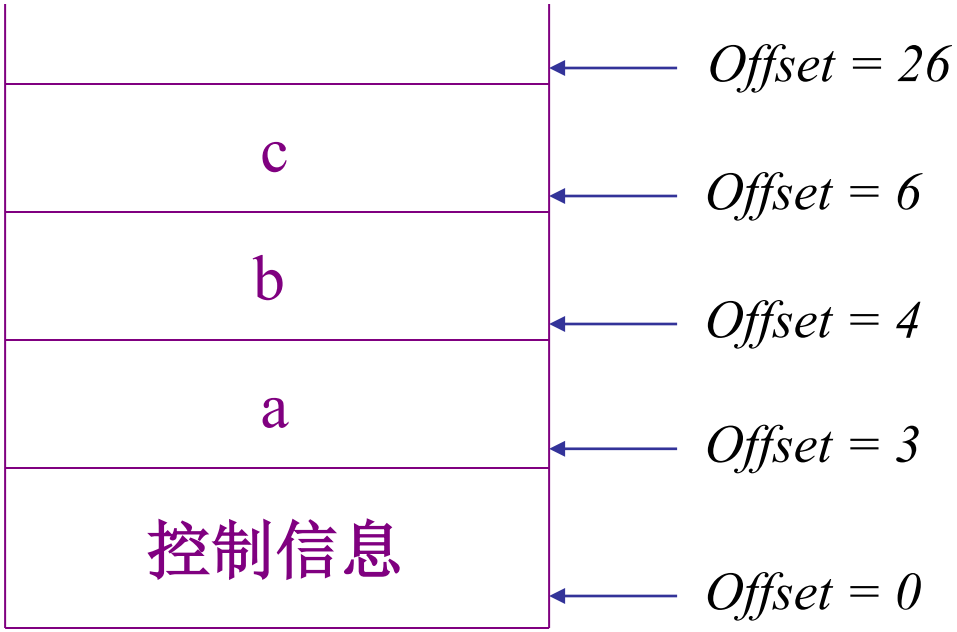


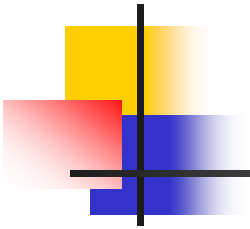
活动记录

– 过程活动记录举例

```
void p( int a)  {
   float b;
   float c[10];
   b=c[a];
}
```

函数 p 的活动记录





活动记录

– 含嵌套过程说明语言的栈式分配

- 主要问题

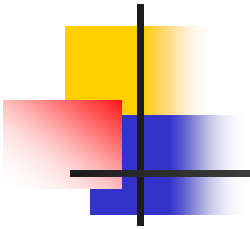
  解决对非局部量的引用（存取）
- 解决方案

  采用 Display 表

  为活动记录增加静态链域

9.3过程调用实现

- 简单过程调用
  - 实在参数的计算和保存
  - 控制转移、返回地址的保存
  - 实在参数和形式参数的结合(多种结合方式)
  - 局部变量的处理
  - 返回值的处理
- 递归过程调用与过程参数
  - 每层过程调用信息的保存与相应信息的查找



小结

- 运行时存储组织概述
- 存储分配策略
  - 静动态分配
  - （栈式、堆式）动态分配
- 活动记录
- 过程调用