

# Operating

System - P134

Part 10

8.2.9 - step 3

## 锁定内存页面

当通过系统调用从文件/设备读取数据到地址空间的缓冲区的进程挂起而此时允许运行的进程产生一个缺页中断

如果使用全局的分页算法，则有很小概率置换包含I/O缓冲区的页面

从而对于正在对该页面进行DMA传输过程的I/O设备

将部分数据写入被置换出的页面，而部分数据写入新装入的页面

一种解决方案是在内存中钉住(pinning)页面

锁住正在进行I/O操作的内存中的页面以保证不被移出内存

另一种解决方案是在内核缓冲区完成所有I/O操作，再将数据复制到用户页面

## 后备存储

在磁盘上设置特殊的交换分区以存放从内存换出的页面，如UNIX从文件系统划分为独立磁盘

这个分区没有普通文件系统，始终使用相应分区的起始块号，以消除文件偏移到块地址的开销

当系统启动时，交换分区为空，在内存中以单独项记录起始位置和大小

一种方案是进程启动时留出与进程大小相同的交换分区

在进程结束后会释放其磁盘上的交换区

每个进程对应其交换区的磁盘地址，即保存进程映像位置

计算返回地址仅将虚拟地址偏移量加到交换区起始地址

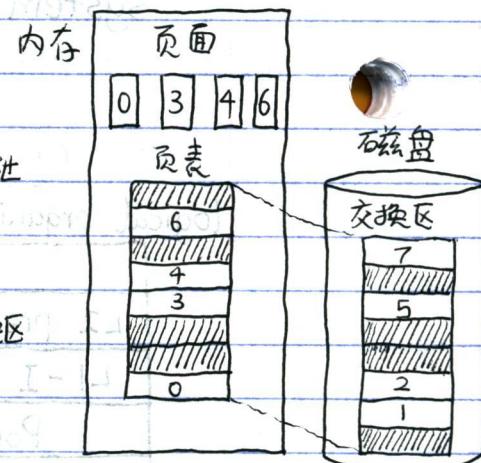
在进程启动前必须初始化交换区

或者将进程映像复制到交换区，或者将进程装入内存

对于增长的数据和堆栈，为正文/数据/堆栈分别保留交换区

计算地址需要进程的分页区域起始位置

由于页面按虚拟页号的顺序连续存储



另一种方案是在进程启动时不作分配

而在页面换出时分配磁盘空间，而换入时收回空间

内存中的进程无需固定于任何交换空间

缺点是内存中每个页面都要记录相应磁盘地址

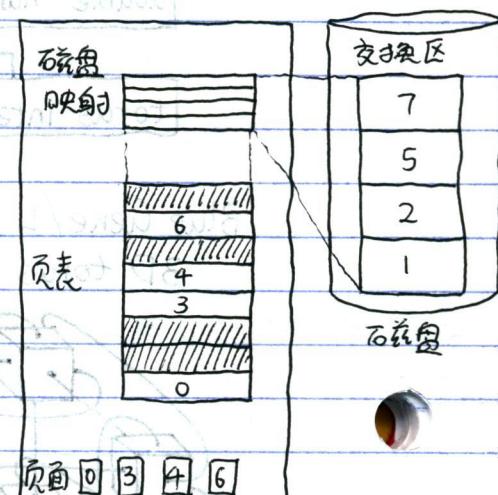
每个进程必须有一张表记录每页面的磁盘位置

页面在磁盘上没有固定地址，内存中的页面没有磁盘副本

页面换出时选择空磁盘页面并据此更新磁盘映射

内存页面在磁盘映射表中的表项

包含非法磁盘地址 / 未被使用的标记位



Windows 在没有磁盘分区可用时，可以利用正常文件系统中较大的事前定位的文件

且由于每个进程的程序正文来自文件系统的文件(可执行)

可以使用这个文件作为交换区

# Operating System - P135

策略与机制分离：通过大多数存储管理器作为用户级进程运行应用于存储管理

Mach 中的存储管理系统分为三个部分

底层 MMU 处理程序：

所有关于 MMU 工作的细节封装在 MMU 处理程序中

代码与机器相关，且操作系统应用于一个新平台时即需重写一次

缺页中断处理程序：

与机器无关的代码，包含大多数分页机制

外部页面调度程序：

由用户空间作为用户进程运行，决定策略

进程启动时，需要通知外部页面调度程序以便建立进程页面映射，并在磁盘分配内存存储

进程运行时，通知外部页面调度程序以将新对象映射到进程地址空间

缺页中断发生时，缺页中断处理程序找出需要的虚拟页面，并向外部页面调度程序发送信息

外部页面调度程序从磁盘中读入所需的页面

将读入页面复制到其自身的地址空间，将位置发送给缺页中断处理程序

缺页中断处理程序从外部页面调度程序的地址空间清楚读入页面的映射

然后请求 MMU 处理程序将其放置到用户地址空间的正确位置

随后重启用户进程

Mach 的实现方案并没有给出放置页面置换算法的位置

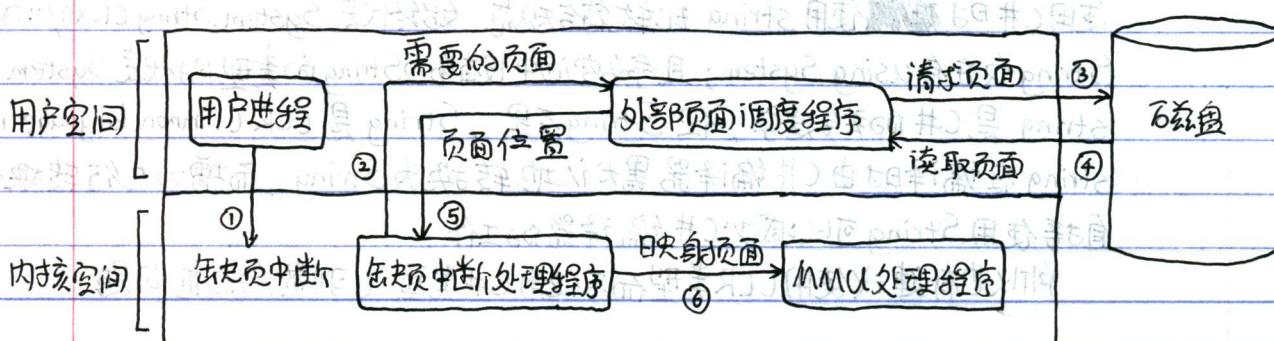
放在外部页面调度程序中，将面对外部页面调度程序无权访问所有页面的 R/W 位

或者通过某种机制发送给外部调度程序，或者将页面置换算法放入内核

后一种中缺页中断处理程序将淘汰页面的数据发送给外部页面调度程序

优势：更多的模块化代码和更好的适应性

缺点：多次交叉“用户-内核”边界和系统模块间消息传递造成的额外开销



# Operating

## System - P136

编译器在编译过程中会建立许多表

用于批处理系统，保存起来供打印清单的源程序正文

符号表，包含变量的名字和属性

常量表，包含所有用到的整型和浮点常量

语法分析树，包含程序语法分析的结果

以上会随着编译进行不断增长

编译器内部过程调用使用的堆栈

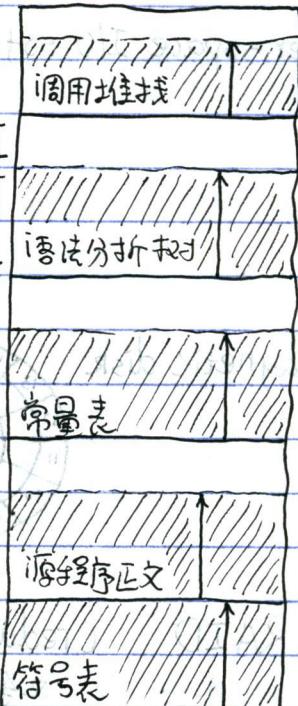
在编译过程中以不可预计的方式增长和缩小

注意当虚拟内存是一维时，虚拟地址从0到最大并紧接上一个

则如果程序变量数量远大于其他部分

则地址空间中分配给符号表的块将被装满

而其他表所在的块中仍有大量空闲空间



### 段 (Segment)

(segment)，机器上提供多个互相独立的地址空间

每个段由一个从0到最大的线性地址序列构成

段长度可以是0到允许最大值之间的任何值，且不同的段长度通常不同，运行期间可动态  
段构成独立的地址空间，为地址增长/减小而不影响到其他的段

如果堆中的堆栈需要更多空间，可以立即得到所需空间

段可能装满，但通常段足够大，于是不会阻挡堆栈增长

段号+段内地址：用于分段/二维存储器中指一个地址

程序员知道段是逻辑实体，并将其作为逻辑实体使用

段可能包含过程/数组/堆栈/数组变量，但不会同时包含不同类型的内容

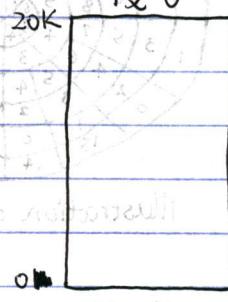
如果每个过程都位于一个独立的段中且起始地址为0

则单独编译的过程的链接操作得到简化

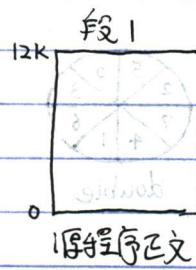
对段n中过程的调用使用二段地址(n, 0)来寻址入0点0

当对段n中的过程修改并重新编译时，不需要对其他过程进行修改

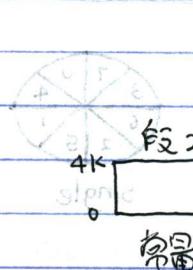
### 段0



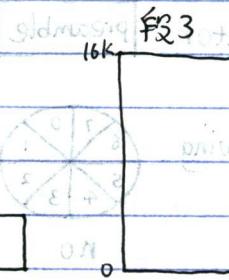
符号表



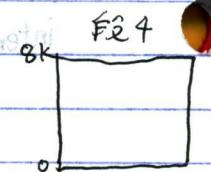
源程序正文



常量



语法分析树



调用堆栈

# Operating

## System - P137

分段

分段有助于在多个进程之间共享过程和数据

常见的例子是共享库 (shared library)

如运行高级窗口系统的现代工作站需要将巨大的图形库编译进几乎所有程序

而分段系统中可以将图形库放入单独的段并由各个进程共享

虽然纯分页系统也可以有共享库，但相对而言更加复杂

且实际上也是通过模拟从分段实现的

由于分段是为程序员所知的逻辑实体，则不同段可以有不同的保护

如过程段可以被指明只允许执行，从而禁止读出和写入

数组段可以被指明允许读写而不允许执行，从而捕获任何试图向此段内的跳转

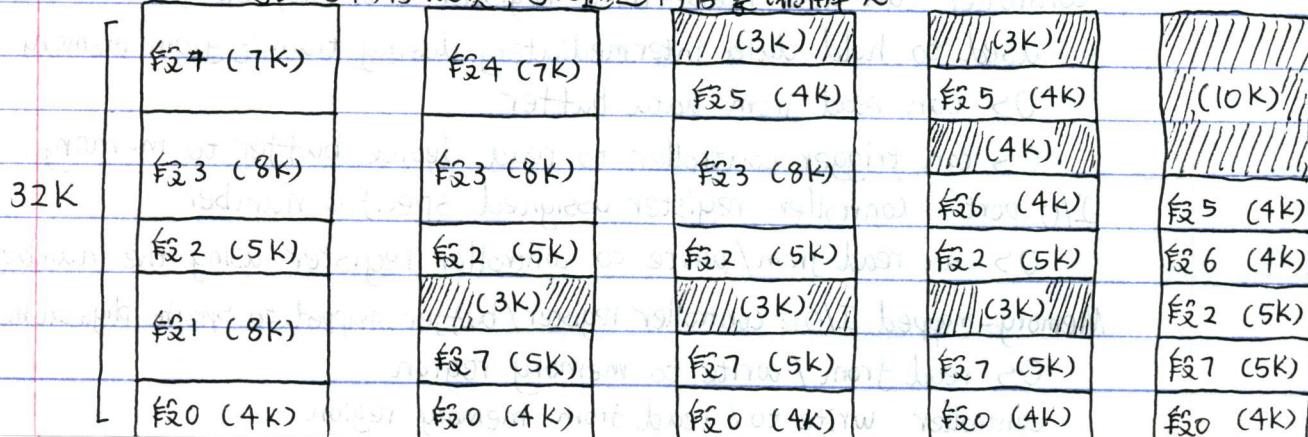
关注点	分页	分段
程序员需要了解技术	否	是
线性地址空间容量	1个	多个
整体地址空间超出物理存储器	可以	可以
区分过程/数据并分别保护	不可以	可以
容易提供大小浮动的表	否	是
方便于用户间过程的共享	否	是
发明技术的目的	为获得更大的线性地址空间 而不必购买更大的物理存储器	使程序/数据可分为 逻辑上独立的地址空间 并有助于共享和保护

外部碎片 (external fragmentation)，又称棋盘形碎片

分段与分页实现本质上的不同：页面是定长的而段不是

当系统运行一段时间后，内存划分为许多块，未包含段的块形成了空闲区

空闲区导致了内存浪费，可以通过内存紧缩解决



# Operating

## System - P138

分段分页结合，当段较大时，将其整体保存在内存中可能不方便甚至是不可能的。  
于是有操作系统实现了对段的分页支持

MULTICS 对 UNIX 系统，x86 存储器体结构，快表以及云计算均有深刻影响

于 1969 年上线，最后一个 MULTICS 系统于 2000 年关闭

几乎没有其他操作系统如 MULTICS 一样几乎没有修改地持续运行了如此长时间

最具创新性的特性是虚拟存储架构

为每个程序提供最多  $2^{18}$  个段

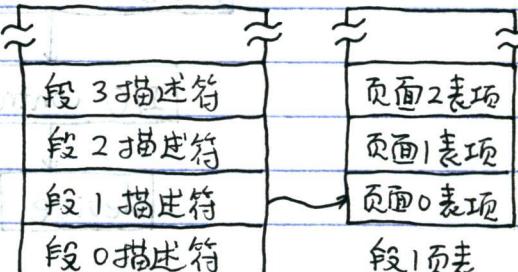
每个段的虚拟地址空间最长为 65536 个字长 (36-bit)

MULTICS 将每个段看作一个虚拟内存并进行分页

分页优点：统一的页面大小

只使用部分段时无需全部调入内存

分段优点：易于编程，模块化，保护，共享



每个 MULTICS 程序维护一个段表，每个段对应一个描述符

由于段表有最多  $2^{18}$  个表项，于是段表本身也作为一个段并分页

段描述符包含一个段是否在内存中的标志

只要段的任何部分在内存中即被认为是在内存中，且页表也在内存中

段描述符包含一个 18 位的指向其页表的指针

由于物理地址是 24-bit 的且页面按 64 字节的边界对齐

于是页面地址的低 6 位为 000000

则描述符只需要 18 位来存储页表地址

段在辅助存储器中的地址不在段描述符中

而是在缺段处理程序使用的另一个表中

每个段都是一个普通的虚拟地址空间

并采用非分段式分页存储方式进行分页

一般页面大小是 1024 个字

35	18	17	9	8	7	6	5	4	3	2	1	0
页表的主存储器地址	段长度						其他位	保护位				

(页面为单位)

页面大小：0: 1024 字 1: 64 字

0: 段是分页的，1: 不分页的

# Operating

## System - P139

MULTICS

在 MULTICS 中地址由段与段内地址构成

段内地址进一步分为页号与页内的字

当 MULTICS 进行内存访问时

- 根据段号查找段描述符

- 检查该段的页表是否在内存中

如果在则查找位置，否则产生段错误

如果违反段保护则发出越界错误(陷阱)

- 检查所请求虚拟页面的页表项

如果该页面不在内存则产生缺页中断

否则从页表项取出页面在内存中的起始地址

- 将偏移量加到页面起始地址以得访问字的内存地址

- 进行内存读/写操作

注意为了简单起见忽略描述符段本身也需要分页的事实

实际过程中通过描述符基址寄存器找到描述符段的页表

这个页表指向描述符段的页面

找到所需段的描述符，即继续进行之后的内存访问步骤

MULTICS 的硬件包含 16 个字的 TLB，保存最近访问的 16 个页面地址

高速 TLB 可以对给定的关键字并行搜索所有表项

当地址传入计算机时，寻址硬件首先检查虚拟地址是否在 TLB 中

如果在则直接从 TLB 中取得页框号并生成要访问的字的实际地址

工作集小于 TLB 容量的进程随着整个工作集装入 TLB 而逐渐达到稳定，否则产生 TLB 错误

比较域 (段号 虚拟页面) 页框 保护 生存时间 表项是否正在使用

<u>并行搜索</u>	6	0	2	只读	10	1
	12	3	1	读/写	2	1
	—	—	—	—	—	0
	2	2	12	只执行	9	1

MULTICS：256K 个为蛇的段，每个段最长有 64K 个 36 位字

X86：16K 个独立的段，每个段最多容纳  $2^{30}$  个 32 位字

较大的段大小，特征比更多的段个数重要得到

自从 X86-64，除了“传统模式”，分段机制已被认为过时且不再被支持

# Operating

## System - P140

x86

虚拟内存的核心是两张表 LDT / GDT

LDT (Local descriptor table, LDT)

每个程序拥有自己的 LDT

描述局部于每个程序的段：代码 / 数据 / 堆栈

GDT (Global descriptor table, GDT)

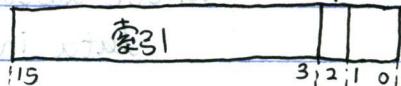
同一台计算机上的所有程序共享同一个 GDT

描述系统段，包括操作系统本身

x86 选择子

0: GDT  
1: LDT

特权级 0~3



当 x86 程序访问段时，必须将这个段的选择子 (selector) 装入机器的一个段寄存器中

CS 寄存器保存代码段的选择子，DS 寄存器保存数据段的选择子

其中一位指明选择子对应的段是局部的 / 全局的

其中 13 位是 LDT/GDT 的表项一编号，于是表长度限制在 8K 个段描述符

描述符 0 是禁止使用的，使用会引起一次陷阱

可以安全地装入一个段寄存器中用于表示这个段寄存器目前不可用

当选择子装入段寄存器时，对应的描述符从 LDT/GDT 中取出并装入微程序寄存器，以便快速访问

由于选择子的合理设计，根据选择子定位描述符很方便

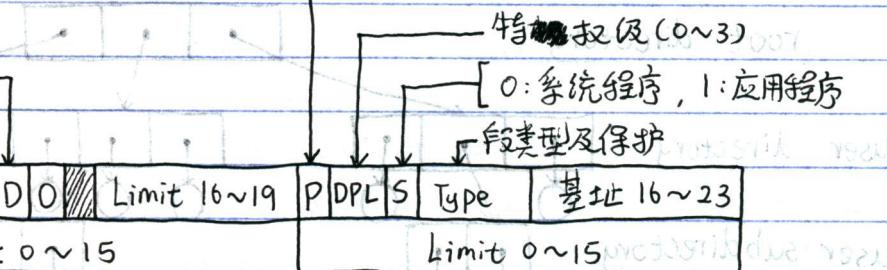
选择 LDT/GDT 后，复制选择子进一个内部擦除寄存器并清除低 3 位

加到 LDT/GDT 表的地址得到直接指向描述符的指针

微程序从内部寄存器中找到对应于选择子的描述符

如果段不存在 (选择子为 0) / 已被换出，则引起一次陷阱

x86 代码段描述符 (数据段不同)



0/1: Limit 以字节 / 页面为单位

描述符 4 基址 24~31 G D O | Limit 16~19 P DPL S Type 基址 16~23

(64-bit) 8-byte 0 基址 0~15 Limit 0~15

硬件根据 Limit (段长度) 域检查偏移量是否超出段的结尾，是则引发一次陷阱

如果 G (粒度) 位域是 0，则精确到字节的段长度，最大为 1MB ( $2^{20}$  byte)

如果是 1，则以页面代替字节为单位给出段的大小

对于 4KB 的页面大小，有最大  $2^{32}$  字节的段长度

(Granule unit size 32 bytes)

# Operating System - P141

分段

X86处理器将描述符中的32-bit基址与偏移量相加

以获得线性地址 (linear address)

基址被分为3片分布在描述符的不同位置

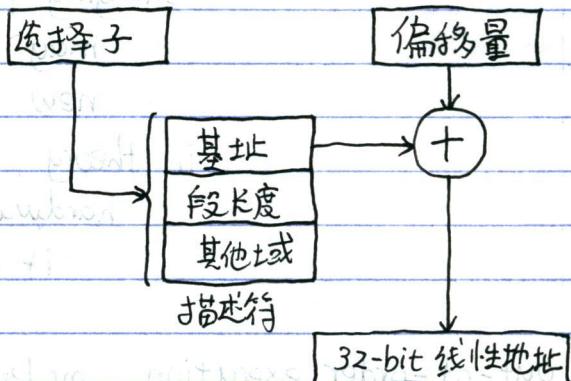
为了和只有24-bit基址的286兼容

通过全局控制寄存器的一位控制允许/禁止分段

禁止分段时，得到一个纯分段方案

线性地址解释为物理地址

并直接传入存储器用于读写操作



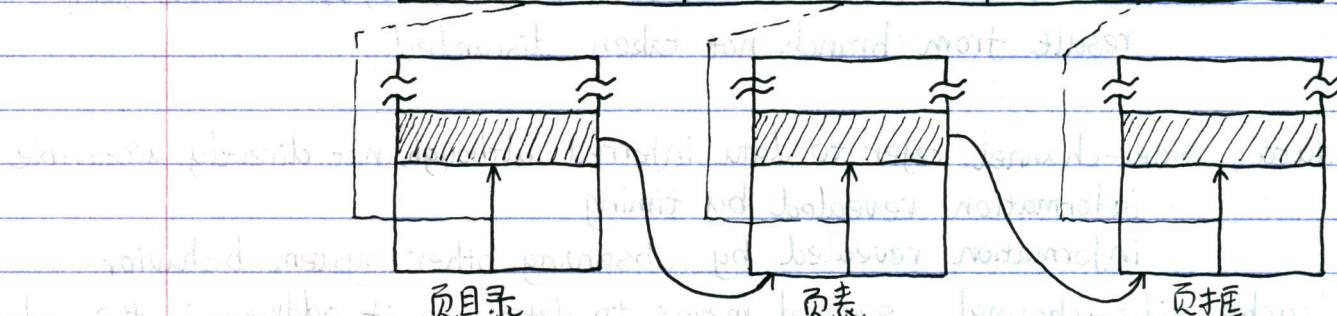
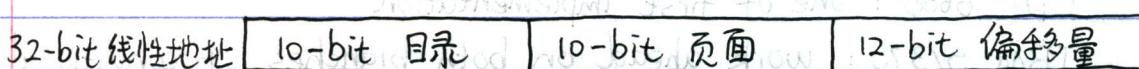
各个段的基址存储在其描述符中

段之间允许覆盖，由于验证段互不重叠过于麻烦费时

允许分页时，线性地址解释为虚拟地址并通过页表映射到物理地址

当使用32-bit虚拟地址和4KB页的情况下较为复杂

单个段可能包含 $2^{20}$ 个页面，于是使用两级映射以减少页表大小



每个程序通过全局寄存器定位一个包含 $2^{10}$ 个32-bit表项的页目录 (page directory)

每个目录项指向一个包含 $2^{10}$ 个32-bit表项的页表，页表项指向页框

32-bit页表项中有20-bit为页框号，其余由硬件设置供操作系统使用的位

一个页表有包含 $2^{10}$ 个4KB页框的表项，于是可以处理4MB的内存

小于4MB的段页目录中只有唯一的表项指向唯一的页表

如果应用程序不需要分段而仅需要一个单独的32-bit地址空间

则使所有段寄存器使用同一个选择子设置，其描述符中基址为0，段长度设置为最大

此时指定偏移量为线性地址，仅使用一个地址空间，效果上等同于正常分页

英特字已经剔除了这个支持了近30年，且表现良好的MULTICS存储模型的变形体