

# 习题课二

## 存储管理

方 钰

---



## 主要知识点:

1. 一般存储管理
2. UNIX V6++ 存储管理





## 主要知识点:

### 1. 一般存储管理

### 2. UNIX V6++ 存储管理



# 主要知识点:

## 1. 一般存储管理

### 可变分区分配

#### 分配算法

##### 首次适应

- 从第1个空闲区开始顺序找, 直至找到一个大小满足要求的空闲分区
- 从该分区中划一块给请求者, 余下的仍留空闲分区表中

矛盾和碎片集中在低地址部分

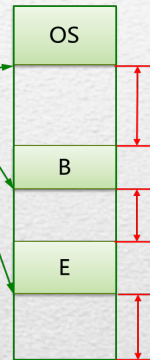
##### 循环首次适应

- 从上次找到空闲分区的下一个空闲分区开始查找 (采用循环查找方式)
- 从该分区中划一块给请求者, 余下的仍留空闲分区表中

缺乏大的空闲分区

#### 按空闲分区首地址升序排列

索引号	大小	起始地址
1	150	1000
2	70	2000
3	100	3000



##### 最佳匹配法

- 总把能满足要求, 又最小的空闲区分配给进程

避免大材小用, 但碎片小

##### 最坏匹配法

- 总把能满足要求, 又最大的空闲区分配给进程, 产生碎片的几率最小。

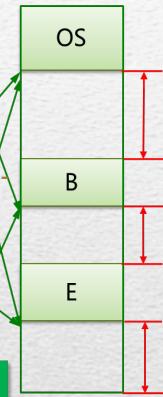
查找效率高, 但缺大分区

#### 按空闲分区大小升序排列

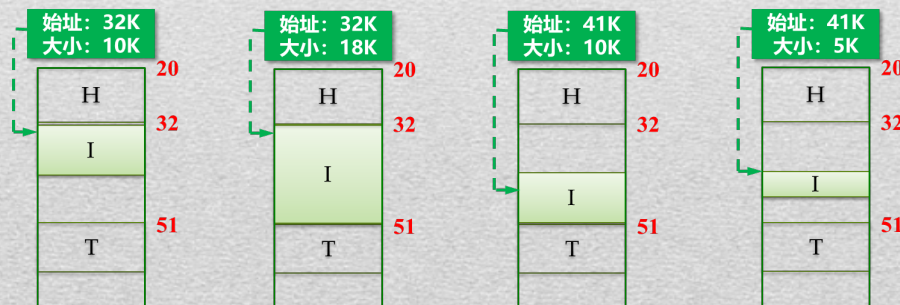
索引号	大小	起始地址
1	70	2000
2	100	3000
3	150	1000

索引号	大小	起始地址
1	150	1000
2	100	3000
3	70	2000

#### 按空闲分区大小降序排列



#### 内存回收



可变分区分配的分配与回收算法: 4种分配算法, 4种回收情况

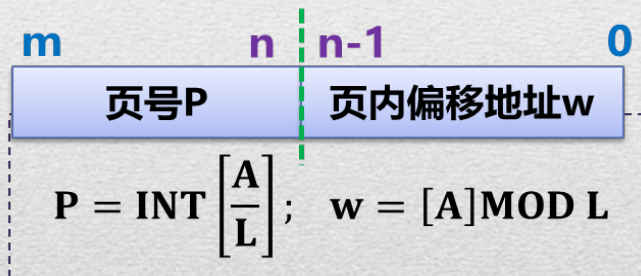


# 主要知识点:

## 1. 一般存储管理

### 程序逻辑地址空间

地址A, 页长:  $L=2^n$   
程序地址分成两个部分:



从0开始顺序编址的程序地址, 称为**线性地址**  
(一维的, 连续的)

例如, 某系统的逻辑页面大小 1KB



分页存储管理中  
逻辑地址与物理  
地址的构成

上例中, 若A = 2170, 则P=2, W=122





# 主要知识点:

## 1. 一般存储管理

### CPU内的两个寄存器

页表长度      页表始址

①页号若大于  
页表长度, 则  
发生越界

②根据页号和页表始址,  
找到对应的页表项

页号      页内偏移地址

程序中的逻辑地址

5      页内偏移地址

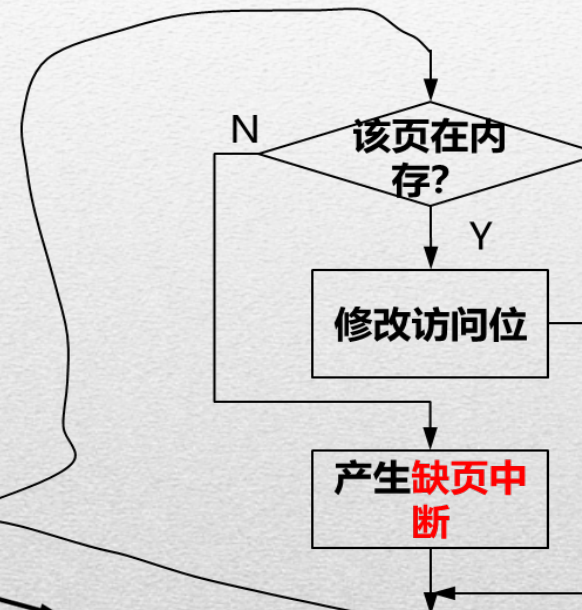
④用页框号代替页号, 获得物理地址

### 页表

	内存页框号	特征位
0	3	XXX
1	5	XXX
2	7	XXX
...	...	XXX

5      XXX

3.5 根据状态位判断该  
页是否在内存



地址变换

页面置换





1. 在下列动态分区分配算法中，最容易产生内存碎片的是

- A. 首次适应算法
- C. 最佳适应算法

- B. 最坏适应算法
- D. 循环首次适应算法

2. 下列措施中，能加快虚实地址转换的是

- I. 增大快表(TLB)容量
- III. 增大交换区(swap)

- II. 让页表常驻内存

A. 仅 I

B. 仅 II

C. 仅 I、II

D. 仅 II、III



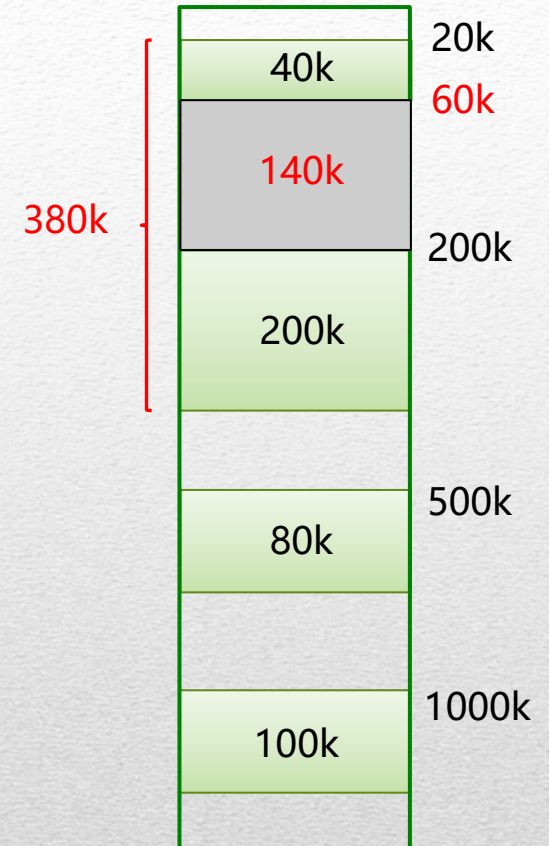


3. 某计算机按字节编址，其动态分区内存管理采用最佳适应算法，每次分配和回收内存后都对空闲分区链重新排序。当前空闲分区信息如下表所示。

分区起始地址	20K	500K	1000K	200K
分区大小	40KB	80KB	100KB	200KB

回收起始地址为60K，大小为140KB的分区后，系统中空闲分区数量、第一个分区的起始地址和大小分别是：

\_\_\_\_\_。







4. 某虚拟存储器用户地址空间有32个页面，每页1KB，主存16KB。假定某时刻为用户的第0, 1, 2, 3号页面分配的物理页号为5, 10, 4, 7, 试将虚拟地址0x0A5C和0x0D3C变化成物理地址。

1KB页长: 低10位为页内偏移地址;  
 32个页面: 虚地址中高5位为页号;  
 主存16KB: 物理地址高4位为页框号。

页表

5
10
4
7

0A5C = 000 1010 0101 1100 虚页号为2, 物理页号为4

000 1010 0101 1100  $\Rightarrow$  01 0010 0101 1100 = 125C

0D3C = 000 1101 0011 1100 虚页号为3, 物理页号为7

000 1101 0011 1100  $\Rightarrow$  01 1101 0011 1100 = 1D3C





5. 某计算机系统按字节编址，采用二级页表的分页存储管理方式，虚拟地址格式如下所示：

页目录号(10位)	页表索引(10位)	页内偏移量(12位)
-----------	-----------	------------

页和页框的大小各为\_\_\_\_，进程的虚拟地址空间大小为\_\_\_\_**页**，假定页目录项和页表项均占4个字节，则进程的页目录和页表最多共占\_\_\_\_页。





6. 有一128行、128列的整数数组A在系统中按行存放。系统采用页式存储管理，内存一个页框可放128个整数。给数组A赋值分别采用程序段（1）、程序段（2）时，各自产生的缺页中断次数为多少。设在内存中给A分配10个物理页面，并且开始时A的第1个页面已在内存。

程序段（1）：

```
for i:=1 to 128
  do for j:=1 to 128
    do A[i][j]:=0;
```

程序段（2）：

```
for j:=1 to 128
  do for i:=1 to 128
    do A[i][j]:=0;
```





7. 在某页式存储管理系统中，一个作业的程序地址空间为3KB，机器内存最大容量为128KB，每个页面的大小为1KB，右表是页表的内容。用户程序中第100号字节单元处有一条指令“MOV ax, (2500)”，该指令的内存地址是\_\_\_\_\_，该指令操作数的内存地址是\_\_\_\_\_。

100字节单元的逻辑地址：

00 00 0110 0100 虚页号为0，查表得页框号为1；

物理地址为：

00 00 0110 0100  $\Rightarrow$  0 0000 0100 0110 0100 = 1124

1
2
4

另一种计算方法：

2500/1024 可知逻辑地址2500在第2号逻辑页，页内偏移地址为452，所以对应的物理地址为4号，页内偏移地址为452，则物理地址为：1024\*4+452=4548





8. 在一个请求分页系统中，采用**LRU页面置换算法**，假如一个作业的页面走向为4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5，当分配给该作业的物理页框数分别为3和4时，试计算访问过程中所发生的缺页次数和缺页率（假设开始的M个页面已装入主存）。

分配给该作业的物理页框数  $M = 3$  时：

	4	3	2	1	4	3	5	4	3	2	1	5
是否缺页				*	*	*	*			*	*	*
内存中包含页面	4	4	4	1	1	1	5	5	5	2	2	2
	3	3	3	3	4	4	4	4	4	4	1	1
	2	2	2	2	2	3	3	3	3	3	3	5
被淘汰的页				4	3	2	1			5	4	3

**缺页7次，缺页率7/12**





8. 在一个请求分页系统中，采用**LRU页面置换算法**，假如一个作业的页面走向为4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5, 当分配给该作业的物理页框数分别为3和4时，试计算访问过程中所发生的缺页次数和缺页率（假设开始的M个页面已装入主存）。

分配给该作业的物理页框数  $M = 4$ 时:

	4	3	2	1	4	3	5	4	3	2	1	5
是否缺页							*			*	*	*
内存中包含页面	4	4	4	4	4	4	4	4	4	4	4	5
	3	3	3	3	3	3	3	3	3	3	3	3
	2	2	2	2	2	2	5	5	5	5	1	1
	1	1	1	1	1	1	1	1	1	2	2	2
被淘汰的页							2			1	5	4

**缺页4次，缺页率4/12**





9. 设某计算机的逻辑地址空间为**64KB**，页面长**4k**。在时刻260前，现运行进程PA的页表如下表所示。系统为每个进程分配3个内存页框；若缺页，采用**FIFO**算法置换。

页号	页框号	存在位	装入时间	上次访问时间	修改位	外存起始扇区号
0	0x100	1	130	250	1	0x2000
1	0x102	1	230	240	0	0x2008
2	-	0	-	-	-	0x2016
3	0x106	1	160	170	0	0x2024

当进程执行到时刻260时，要访问逻辑地址为0x27CA的数据，计算该逻辑地址对应的物理地址。





0x27CA: 该逻辑地址对应的页号是2, 页内偏移地址是0x7CA

页号	页框号	存在位	装入时间	上次访问时间	修改位	外存起始扇区号
0	0x100	1	130	250	1	0x2000
1	0x102	1	230	240	0	0x2008
2	-	0	-	-	-	0x2016
3	0x106	1	160	170	0	0x2024

由逻辑页号2查表可知, 该逻辑页不在内存。会引发缺页中断, 缺页中断的处理过程如下:





- ① (选换出页) 由于采用FIFO算法, 可知选择选择0号页换出;
- ② (回写脏页) 由于0号页在内存中曾经被修改过, 因此需启动I/O操作将0号页写回磁盘上0x2000开始的8个盘块中;
- ③ (读入新页) 启动I/O将0x2016开始的8个盘块的内容读入0x100号物理页框;
- ④ (修改页表) 修改页表如下:

页号	页框号	存在位	装入时间	上次访问时间	修改位	外存起始扇区号
0	-	0	-	-	-	0x2000
1	0x102	1	230	240	0	0x2008
2	0x100	1	260	260	0	0x2016
3	0x106	1	160	170	0	0x2024

- ⑤ (形成地址)  $0x27CA \Rightarrow 0x1007CA$





## 主要知识点:

### 1. 一般存储管理

### 2. UNIX V6++存储管理

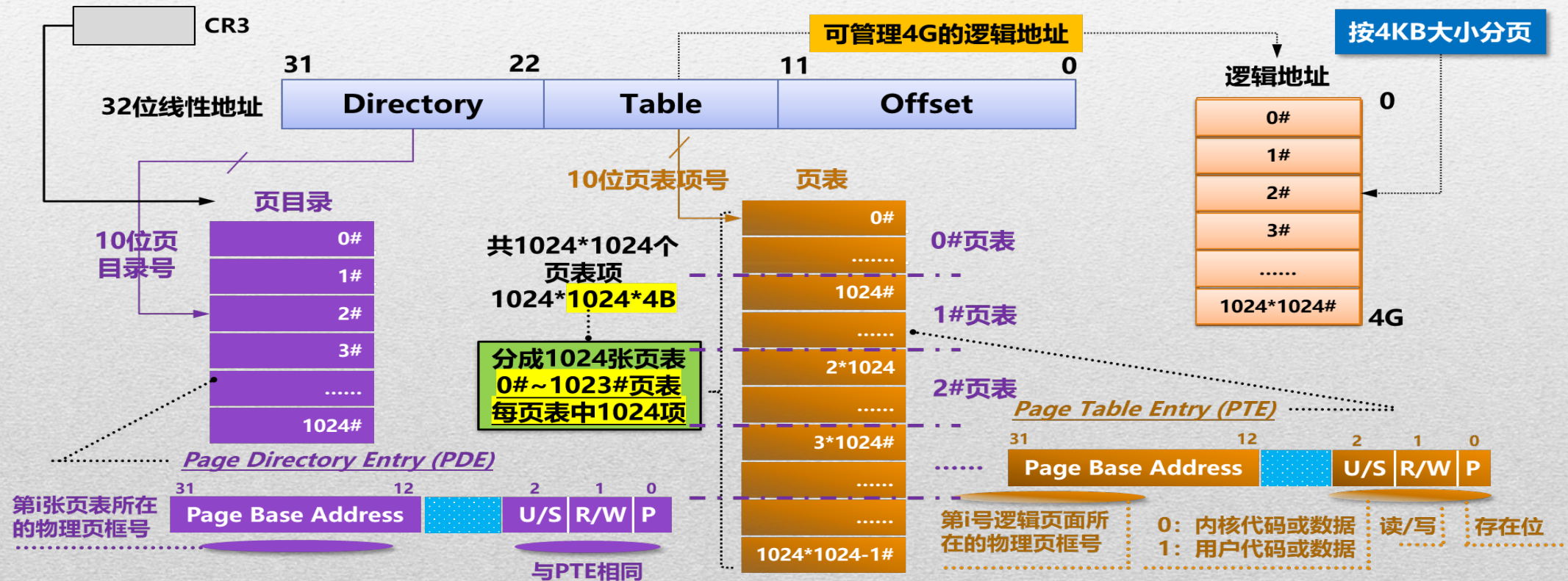


## 主要知识点:

## 2. UNIX V6++ 存储管理

## i386线性地址空间

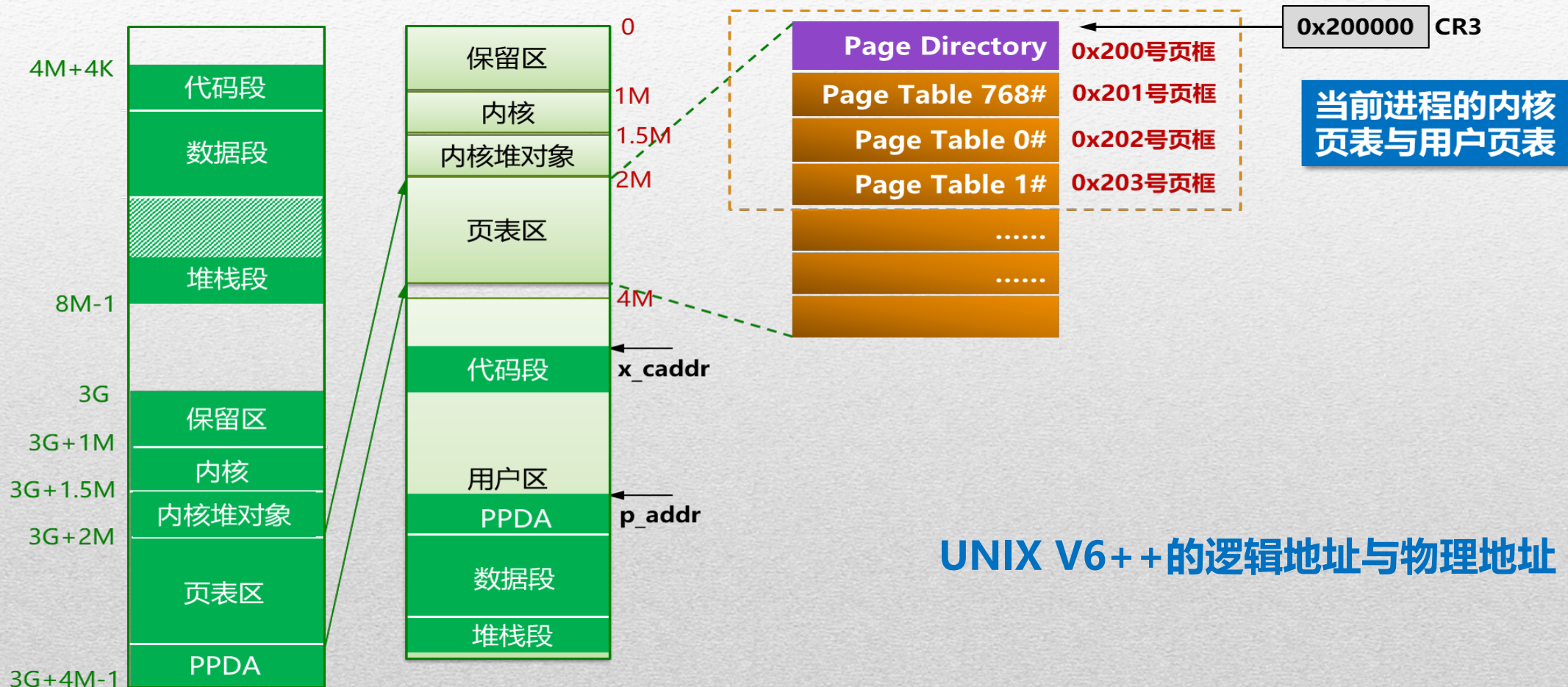
## 使用二级页表管理进程的线性地址





# 主要知识点:

## 2. UNIX V6++ 存储管理





## 主要知识点:

## 2. UNIX V6++ 存储管理

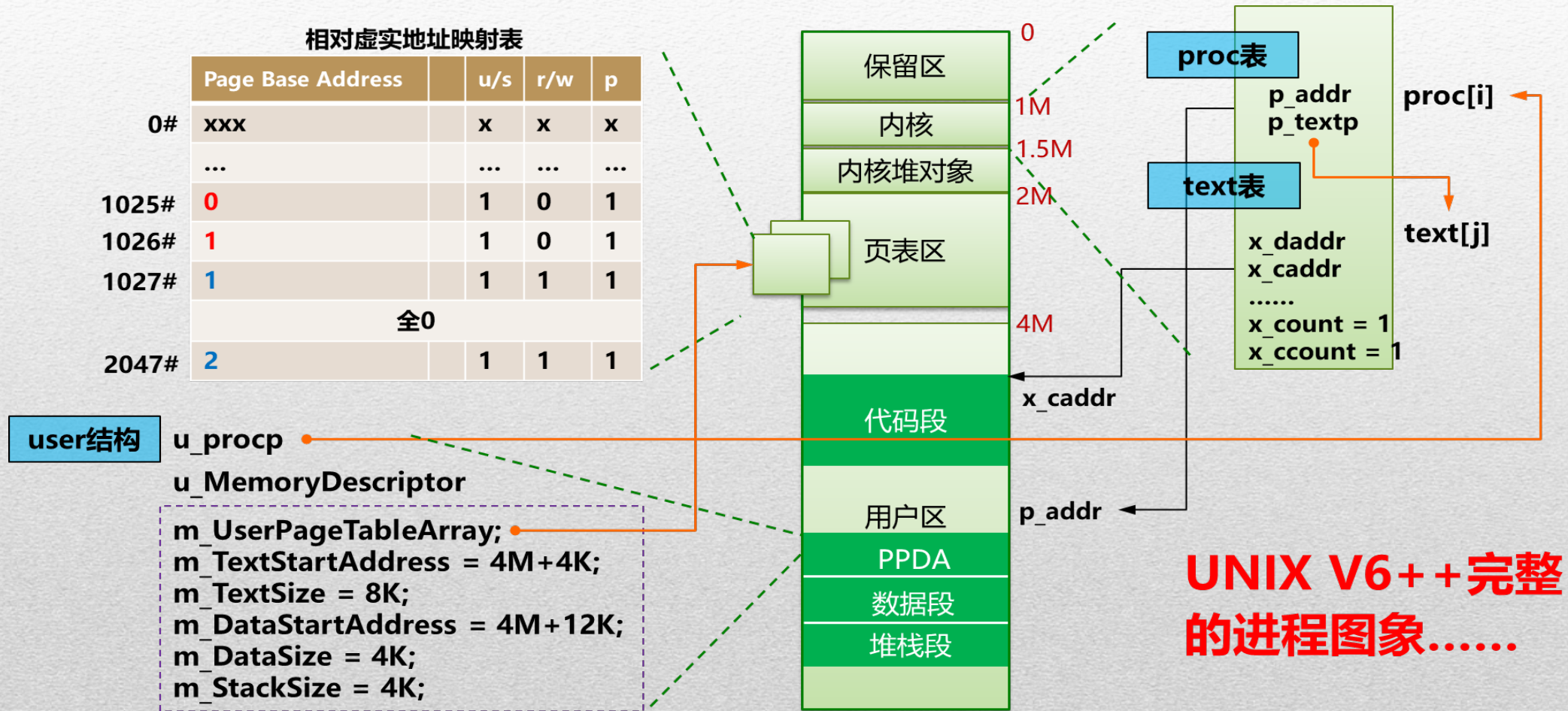






# 主要知识点:

## 3. UNIX V6++存储管理







	名称	类型	含义
进程标识	p_uid	short	用户ID
	p_pid	int	进程标识数，进程编号
	p_ppid	int	父进程标识数
进程图象在内存中的位置信息	p_addr	unsigned long	ppda区在物理内存中的起始地址
	p_size	unsigned int	进程图象（除代码段以外部分）的长度，以字节单位
	p_textp	Text *	指向该进程所运行的代码段的描述符
进程调度相关信息	p_stat	ProcessState	进程当前的调度状态
	p_flag	int	进程标志位，可以将多个状态组合
	p_pri	int	进程优先数
	p_cpu	int	cpu值，用于计算p_pri
	p_nice	int	进程优先数微调参数
	p_time	int	进程在盘交换区上（或内存内）的驻留时间
	p_wchan	unsigned long	进程睡眠原因
信号与控制台终端	p_sig	int	进程信号
	p_ttyp	TTY*	进程tty结构地址





# Process类

进程图象在内存中的位置信息

p_addr	unsigned long	ppda区在物理内存中的起始地址
p_size	unsigned int	进程图象（除代码段以外部分）的长度，以字节单位
p_textp	Text *	指向该进程所运行的代码段的描述符

# Text类

位置相关

名称	类型	含义
x_daddr	int	代码段在盘交换区上的地址
x_caddr	unsigned long	代码正文段在物理内存中的起始地址，以字节为单位
x_size	unsigned int	代码段长度，以字节为单位
*x_iptr	Inode	内存inode地址（用于相应的可执行文件的管理）

共享代码的进程数

x_count	unsigned int	共享该代码段的进程数
x_ccount	unsigned short	共享该代码段，且图像在内存的进程数





# User类

	名称	类型	含义
进程的用户标识	u_uid	short	有效用户ID
	u_gid	short	有效组ID
	u_ruid	short	真实用户ID
	u_rgid	short	真实组ID
进程的时间相关	u_utime	int	进程用户态时间
	u_stime	int	进程核心态时间
	u_cutime	int	子进程用户态时间总和
	u_cstime	int	子进程核心态时间总和
现场保护相关	u_rsav[2]	unsigned long	用于保存esp与ebp指针
	u_ssav[2]	unsigned long	用于对esp和ebp指针的二次保护
内存管理相关	*u_procp	Process	指向该u结构对应的Process结构 (逻辑地址)
	u_MemoryDescriptor	MemoryDescriptor	封装了进程图象在逻辑内存中的位置、大小等信息
系统调用相关	EAX = 0	static const int	访问现场保护区中EAX寄存器的偏移量
	*u_ar0	unsigned int	指向核心栈现场保护区EAX寄存器存放的栈单元
	u_arg[5];	int	存放当前系统调用参数
	*u_dirp	char	系统调用参数 (一般用于Pathname) 的指针





10. UNIX V6++系统中，进程pa的代码段长度为8k字节，数据段长度为8k字节。请回答下列问题：

(1) 内核在逻辑地址**3G+2M+48K**处为该进程创建相对虚实地址映射表。画出该进程的相对虚实地址映照表，并填内存描述符u\_MemoryDescriptor中各参数的值。

相对虚实地址映射表

```

PageTable*    m_UserPageTableArray =
unsigned long m_TextStartAddress =
unsigned long m_TextSize =
unsigned long m_DataStartAddress =
unsigned long m_DataSize =
unsigned long m_StackSize =
  
```

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	...		...	...	...
1024#	xxx		x	x	x
1025#					
1026#					
1027#					
2047#					





10. UNIX V6++系统中，进程pa的代码段长度为8k字节，数据段长度为8k字节。请回答下列问题：

(2) 如果该进程为现运行进程，且该进程的p\_addr=8M+256K，x\_caddr= 8M+128K，请尽量详细地绘制出该进程的图象和4张页表的内容。

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	...		...	...	...
1024#	xxx		x	x	x
1025#					
1026#					
1027#					
2047#					

Page Directory (0x200号页框)

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	.....				
768#	0x201		0	1	1
	.....				

	PBA		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#					

Page Table 0# (0x202号页框)

	PBA		u/s	r/w	p
	/		/	/	/

Page Table 1# (0x203号页框)

	PBA		u/s	r/w	p
0#	/		/	/	/
1#					
1023#					





10. UNIX V6++系统中，进程pa的代码段长度为8k字节，数据段长度为8k字节。请回答下列问题：

(3) 是否可以不用相对虚实地址映射表？

①.  $p\_addr \gg 12$ , 写入

```

PageTable* m_UserPageTableArray = 3G+2M+48K;
unsigned long m_TextStartAddress = 4M+4K;
unsigned long m_TextSize = 8K;
unsigned long m_DataStartAddress = 4M+12K;
unsigned long m_DataSize = 8K;
unsigned long m_StackSize = 4K;
  
```

```

②. for (i = 0; i < 代码段长度/4K; i++)
{
    1#页表中代码段起始位置的PBA =  $x\_caddr \gg 12 + i$ ;
    设置标志位;
}
  
```

Page Table 768# (0x200号页框)

	PBA		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x600		0	1	1

Page Table 1# (0x203号页框)

	PBA		u/s	r/w	p
0#	/		/	/	/
1#	0x820		1	0	1
	0x821		1	0	1
	0x841		1	1	1
	0x842		1	1	1
	.....		...	...	...
1023#	0x843		1	1	1





10. UNIX V6++系统中，进程pa的代码段长度为8k字节，数据段长度为8k字节。请回答下列问题：

(3) 是否可以不用相对虚实地址映射表？

①.  $p\_addr \gg 12$ , 写入

```

PageTable* m_UserPageTableArray = 3G+2M+48K;
unsigned long m_TextStartAddress = 4M+4K;
unsigned long m_TextSize = 8K;
unsigned long m_DataStartAddress = 4M+12K;
unsigned long m_DataSize = 8K;
unsigned long m_StackSize = 4K;
  
```

Page Table 768# (0x200号页框)

	PBA		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x600		0	1	1

Page Table 1# (0x203号页框)

	PBA		u/s	r/w	p
0#	/		/	/	/
1#	0x820		1	0	1
	0x821		1	0	1
	0x841		1	1	1
	0x842		1	1	1
	.....		...	...	...
1023#	0x843		1	1	1

```

③. for (i = 0; i < 数据段长度/4K; i++)
{
    1#页表中数据段起始位置的PBA =  $p\_addr \gg 12 + i + 1$ ;
    设置标志位;
}
  
```