# 进程控制与进程间通信

北京邮电大学计算机学院 蒋砚军
jiangyanjun0718@bupt.edu.cn

# 进程的基本概念和逻辑内存

# 进程与程序

## 程序

- 指令和数据的集合
- 存放在磁盘上的一个普通文件里
- 文件的i节点中标为可执行，内容符合系统要求

## 进程

- 包括指令段、用户数据段和系统数据段的执行环境

## 进程和程序的关系

- 程序用于初始化进程的指令段和用户数据段，初始化后，进程和初始化它的程序之间无联系
- 进程运行时磁盘上的程序文件不可修改/删除
- 同时运行的多个进程可由同一程序初始化得到，进程之间没什么联系。内核通过安排它们共享指令段甚至不同程序的进程共享函数库（动态链接）以节省内存，但这种安排对用户来说是透明的

# 进程的组成部分(1)

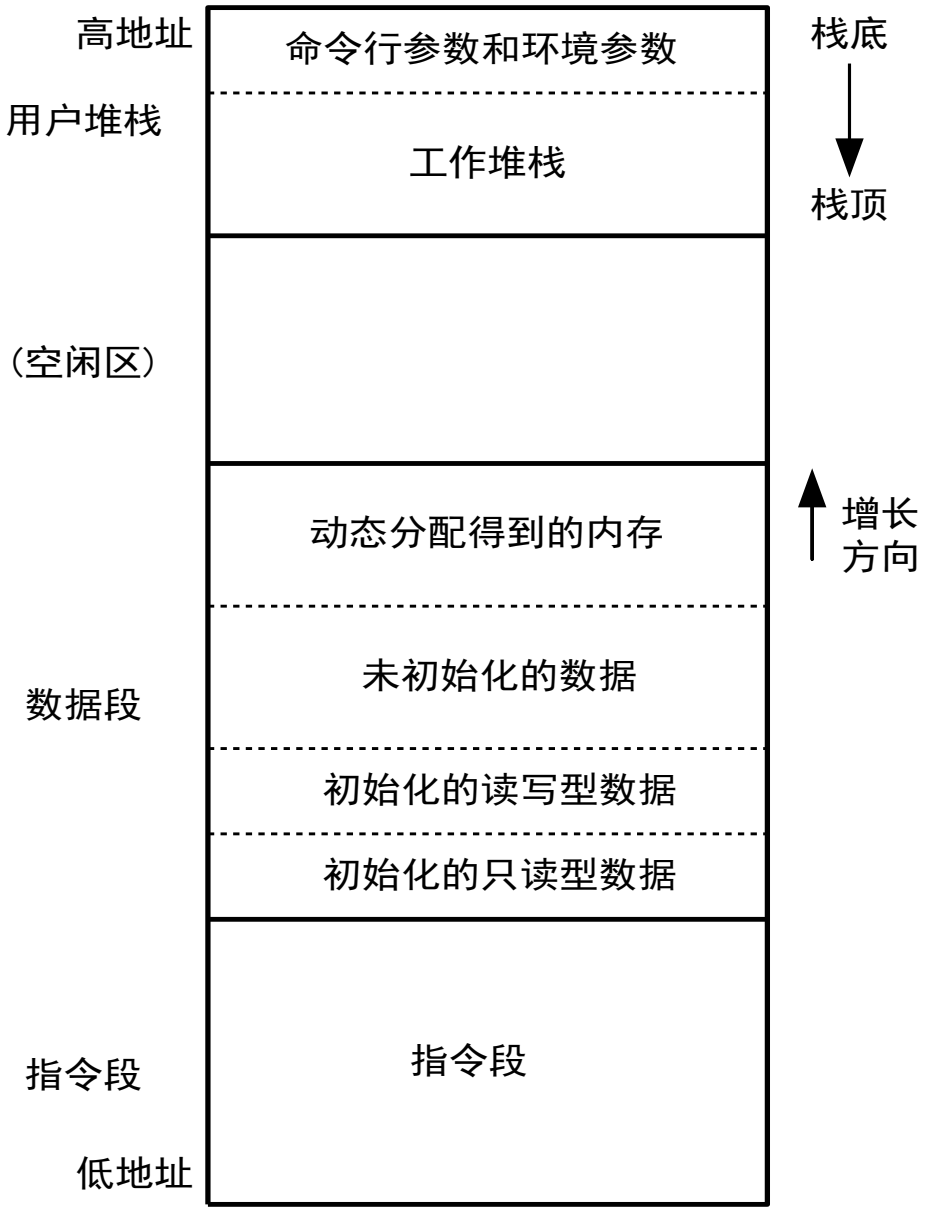
## 四部分：指令段，数据段，栈段和系统数据

- **指令段(Text)**
  - 程序的CPU指令代码,包括：主程序和子程序编译后的CPU指令代码，以及调用的库函数代码
  - 指令段的大小固定不变，只读
- **用户数据段**
  - 全局变量，静态(static)变量，字符串常数
  - 允许数据段增长和缩小，实现内存的动态分配
    - 系统调用sbrk()允许编程调整数据段的大小（调整单位为“页”）
    - 内存管理库函数，如：malloc()，free()

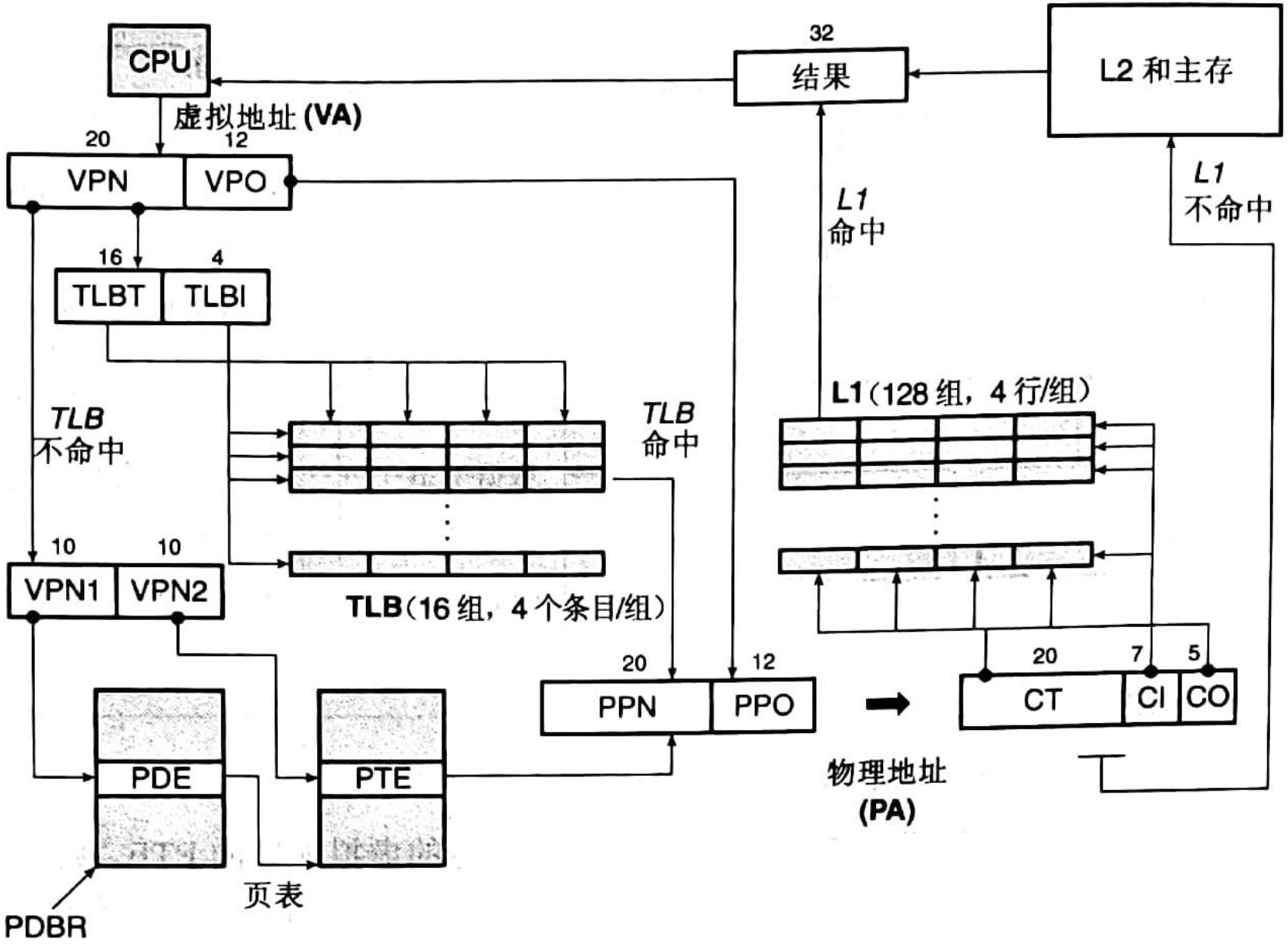
# 进程的组成部分(2)

- **用户栈段**
  - 程序执行所需要的栈空间，实现函数的调用
    - 用于保存子程序返回地址
    - 在函数和被调函数之间传递参数
    - 函数体内部定义的变量(静态变量除外)
  - main函数得到的命令行参数以及环境参数
    - 存放在栈的最底部
    - main函数运行之前，这些部分就已经被系统初始化
  - 栈段的动态增长与增长限制
- **系统数据段**
  - 上述三部分在进程私有的独立的逻辑地址空间内（CPU用户态访问）
  - 系统数据段是内核内的数据，每个进程对应一套
    - 包括页表和进程控制块PCB

# 进程虚拟地址空间的布局



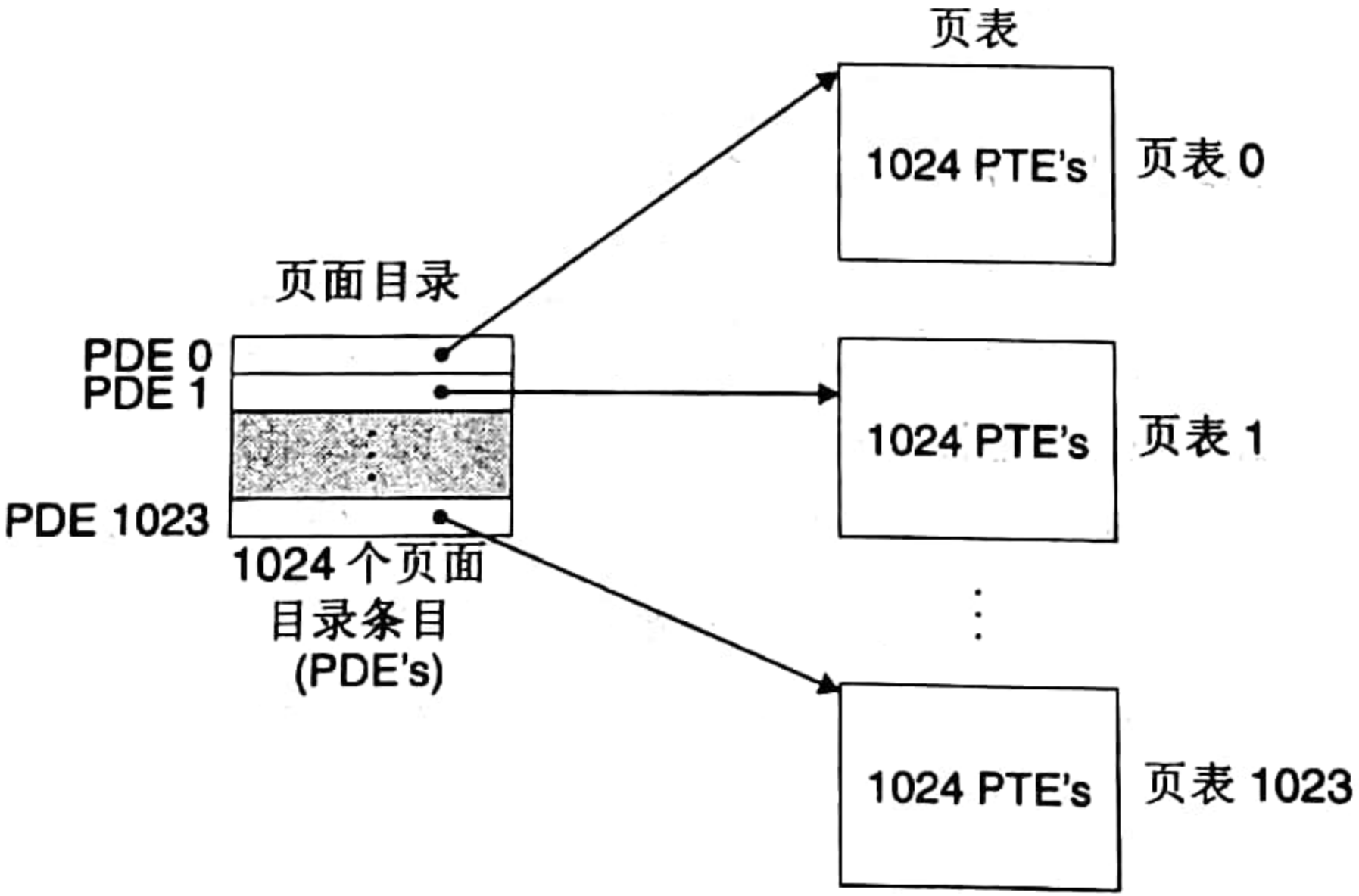
# Pentium虚拟地址到物理地址转换



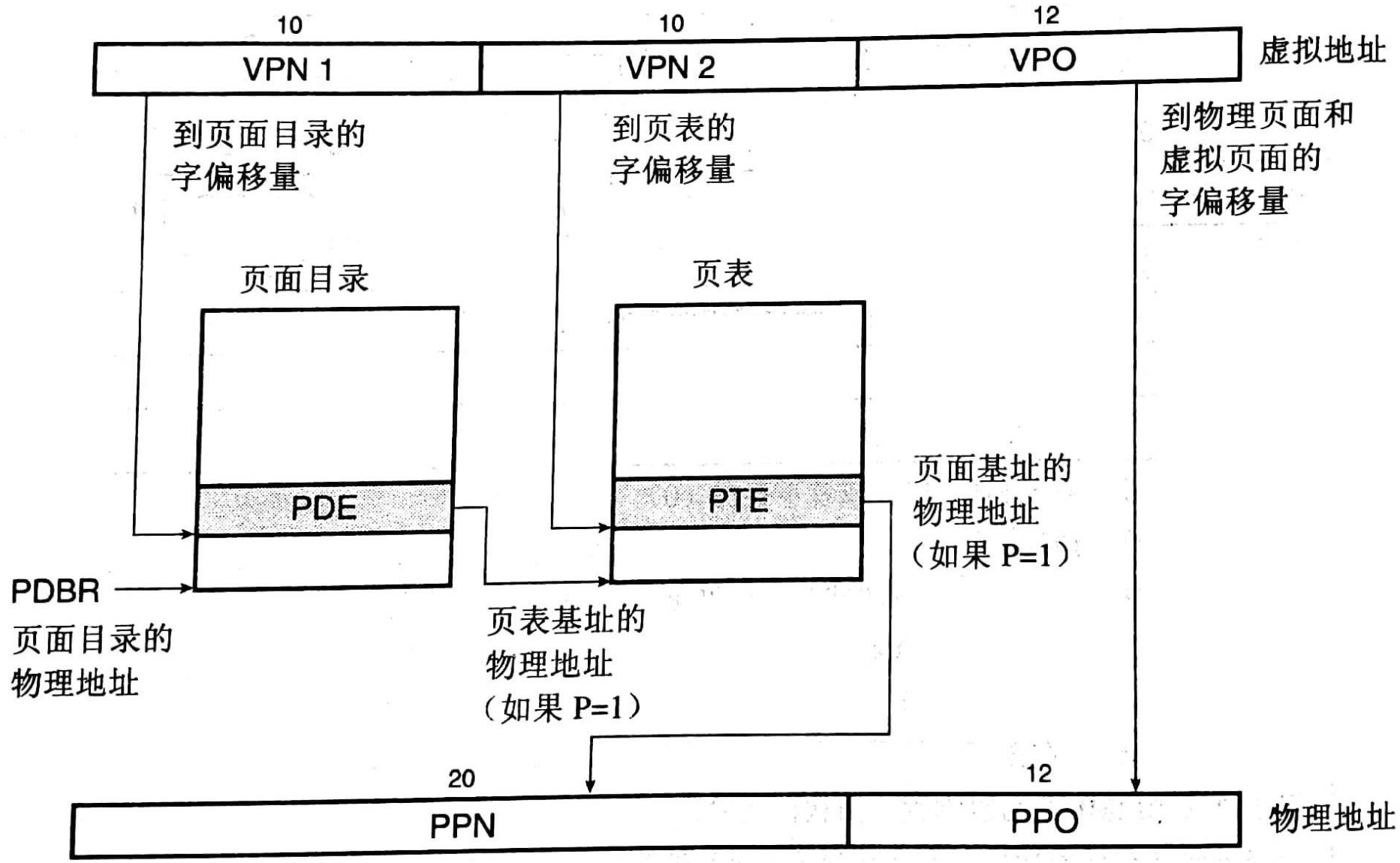
# 术语

| | | |
|---|---|---|
| VA | 虚拟地址 | Virtual Address |
| VPN | 虚拟页号 | Virtual Page Number |
| VPO | 虚拟页内偏移量 | Virtual Page Offset |
| PA | 物理地址 | Physical Address |
| PPN | 物理页号 | Physical Page Number |
| PPO | 物理页内偏移量 | Physical Page Offset |
| PD | 页面目录 | Page Directory |
| PDE | 页面目录条目 | Page Directory Entry |
| PDBR | 页面目录基地址寄存器 | Page Directory Base Register |
| PT | 页表 | Page Table |
| PTE | 页表条目 | Page Table Entry |
| TLB | 地址翻译缓冲（快表） | Translation Lookaside Buffer |
| TLBI | TLB索引 | TLB Index |
| TLNT | TLB标记 | TLB Tag |

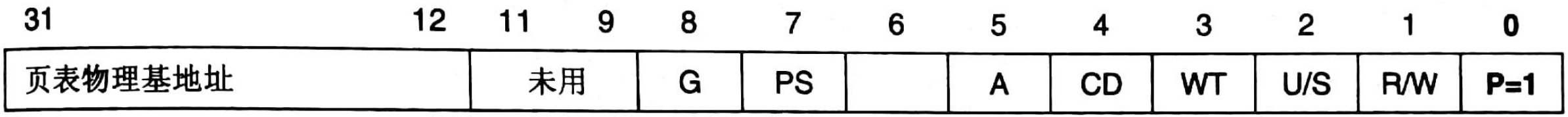
# Pentium的两级页表



# Pentium的虚实地址转换



# PDE




| 字段 | 描　述 |
|---|---|
| P | 页表存在于物理存储器中（1）或者不存在（0） |
| R/W | 只读或者读-写访问许可 |
| U/S | 用户或者超级用户模式（内核模式）访问许可 |
| WT | 对这个页表的直写或者写回缓存策略 |
| CD | 缓存禁止（1）或者启用（0） |
| A | 这个页被访问过吗？（在读写时由MMU设置，由软件清除） |
| PS | 页面大小为4KB（0）或者4MB（1） |
| G | 全局页面（在任务切换时，不会从TLB中驱除掉） |
| PT 基地址 | 物理页表地址的最高 20 位 |

# PTE

| 31 12 | 11 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 页面物理基地址 | 未用 | G | O | D | A | CD | WT | U/S | R/W | P=1 |

| | |
|---|---|
| 操作系统可用的（在二级存储中的页面位置） | P=0 |

| 字段 | 描　　述 |
|---|---|
| P | 页表存在于物理存储器中（1）或者不存在（0） |
| R/W | 只读或者读/写访问许可 |
| U/S | 用户或者超级用户模式（内核模式）访问许可 |
| WT | 对这个页表的直写或者写回缓存策略 |
| CD | 缓存禁止（1）或者启用（0） |
| A | 引用位（由MMU在读写时设置，由软件清除） |
| D | 修改位（由MMU在写时设置，由软件清除） |
| G | 全局页面（在任务切换时不会从TLB中驱除掉） |
| 页面基地址 | 物理页表地址的最高 20 位 |

# 能解决的问题

- 内存越界
- 内存读写保护
- 缺页调入
- 利用磁盘交换区实现以较小物理内存提供较大逻辑内存的效果
- 栈段的生长
- 写时复制
- LRU算法判断页有没有被访问
- 淘汰页面时判断是否需要回写磁盘

# 进程的执行状态

# 进程的系统数据

在UNIX内核中，含有进程的属性，包括:

- 页表
- 进程状态，优先级信息
- 核心堆栈
- 当前目录(记录了当前目录的i-节点)，根目录
- 打开的文件描述符表
- umask值
- 进程PID，PPID
- 进程主的实际UID/GID，有效UID/GID
- 进程组组号

# 传统UNIX的user+proc结构

## 进程PCB被分为user结构和proc结构两部分

- user结构(约5000字节)，<sys/user.h>
  - 进程运行时才需要的数据在user结构
  - 核心态堆栈占用了较多空间
- proc结构(约300字节)，<sys/proc.h>
  - 进程不运行时也需要的管理信息存于proc结构
- 用户程序不能直接存取和修改进程的系统数据
- 系统调用可用来访问或修改这些属性
  - chdir， umask，open， close，setpgrp， getpid， getppid

# 进程的基本状态

## 基本状态

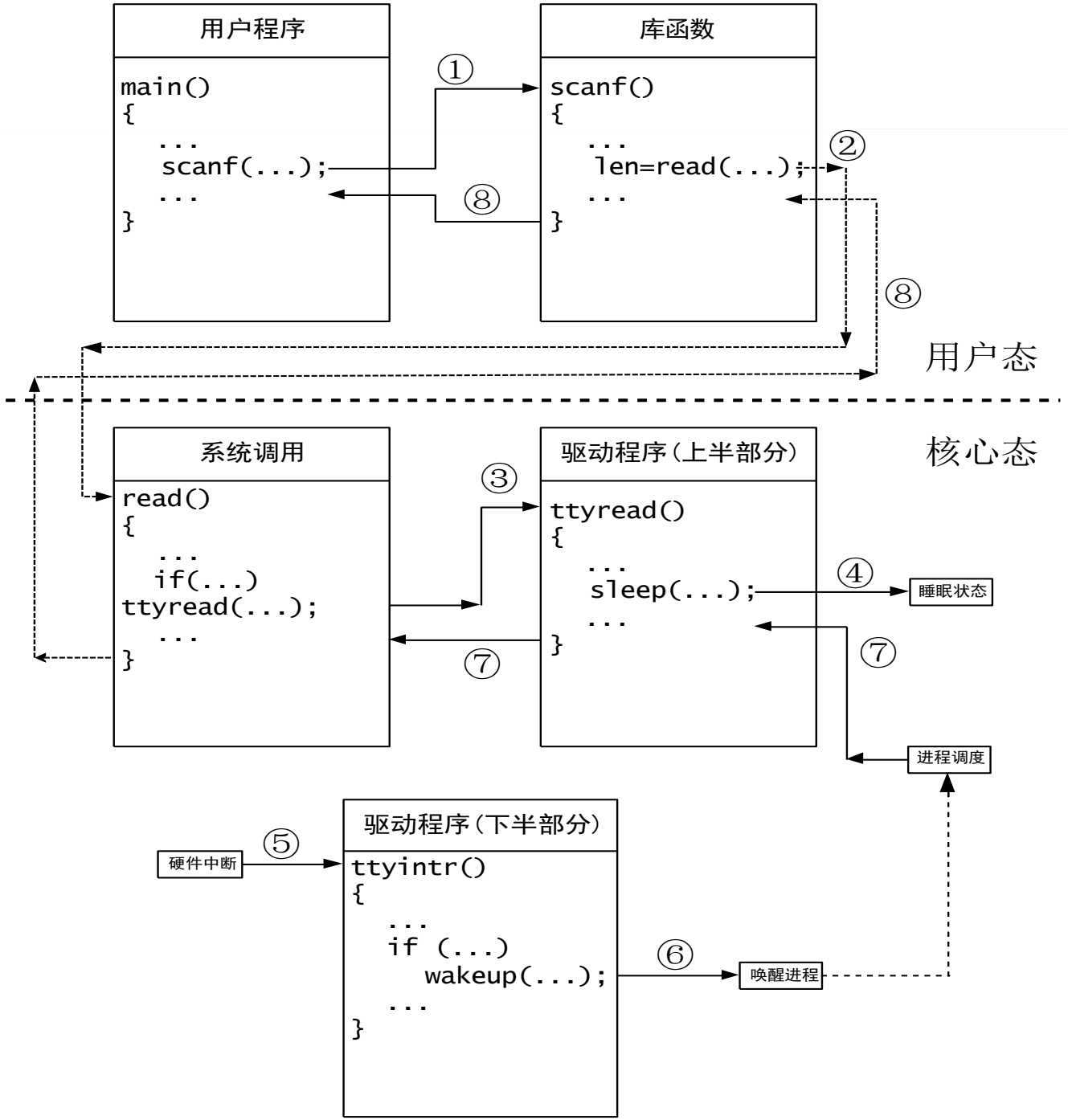
- 进程创建之后，主要有运行状态和睡眠状态(也叫阻塞状态，等待状态，挂起状态，等等)
- 内核总是在分时处理运行状态的进程，而不顾那些处于睡眠状态的进程
- 睡眠状态的进程，在条件满足后转化为运行状态
- 进程在睡眠时，不占用CPU时间
  - **注意：在编程时，尽量不要让程序处于忙等待状态**

# 进程的调度

## 调度优先级

- 内核将可运行进程按优先级调度，高优先级进程优先
- 进程的优先级总在不停地发生变化
- 处于睡眠状态的进程一旦被叫醒后，被赋以高优先级，以保证人机会话操作和其它外设的响应速度
- 用户程序用nice()系统调用有限地调整进程的优先级

# 进程进入阻塞态，然后恢复运行




**scanf("%d", &n);**

**程序在串口终端上执行时操作员输入756然后按下回车**

# 命令ps

## 功能

- 查阅进程状态(process status)(实际上就是将内核中PCB数组的内容有选择地打印出来)

## 选项

- 用于控制列表的行数(进程范围)和列数(每进程列出的属性内容)
- 无选项：只列出在当前终端上启动的进程
  - 列出的项目有：PID，TTY，TIME，COMMAND
- **e**选项：列出系统中所有的进程(进程范围)
- **f**选项：以full格式列出每一个进程(控制列的数目)
- **l**选项：以long格式列出每一个进程(控制列的数目)

# 命令ps举例

## 例：ps -ef命令的输出

```
 UID   PID  PPID  C    STIME  TTY TIME COMMAND
root     0     0  0 16:11:14  ?   0:00 sched
root     1     0  0 16:11:14  ?   0:01 /etc/init
root     2     0  0 16:11:14  ?   0:00 vhand
root     3     0  0 16:11:14  ?   0:00 bdflush
root  6547   335  0 16:15:05  01  0:00 server 4 1
root   175     1  0 16:11:14  ?   0:00 inetd
root   173     1  0 16:11:14  ?   0:00 syslogd
root   296     1  0 16:11:21  ?   0:00 /tcb/files/no_luid/sdd
root  6551   291 80 16:17:16  08  0:37 busy_check
root   335     1  0 16:11:31  01  0:00 server_ctl
root   337     1  0 16:11:33  01  0:05 server_ap
root   353     1  0 16:12:01  12  0:00 client_ctl 100.1.1.3
root   356   295  0 16:12:07  12  0:04 client_ap 1403
```

# 命令ps列出的进程属性

- UID：用户ID(注册名)
- PID：进程ID
- **C**：CPU占用指数：最近一段时间(秒级别)进程占用CPU情况。不同系统算法不同，例如：正占CPU进程10ms加1，所有进程1秒衰减一半
- PPID：父进程的PID
- STIME：启动时间
- **SZ**：进程逻辑内存大小(Size)
- TTY：终端的名字
- COMMAND：命令名
- WCHAN：进程睡眠通道(Wait Channel)，进程阻塞在何处
- **TIME**：累计执行时间(占用CPU的时间)
- PRI：优先级
- **S**：状态，S(Sleep)，R(Run)， Z(Zombie)

# 进程的执行时间

# time:进程执行的时间

- 进程执行时间包括
  - 睡眠时间，CPU时间(用户时间和系统时间)
  - 外部命令/usr/bin/time和内部命令time输出的数据格式不同

/usr/bin/time find /usr -name '*.c' -print

Real 6.06
User 0.36
System 2.13

- C-shell: time find /usr -name '*.h' -print

0.4u 6.2s 0:10 61% 4+28k 0+0io 0pf+0w

- 与CPU时间有关的命令vmstat

```
$ vmstat 10
   procs                  memory      swap        io     system      cpu
 r  b  w  swpd   free   buff   cache  si   so    bi  bo   in       cs us sy id
 0  0  0     0  55916   6128  38156  0    0   439 118  146   180  8 15 76
 0  0  0     0  55252   6160  38160  0    0     0  32  112    54  26  1 73
```

# 系统调用times()

## 系统调用times()

当前进程CPU时间，已结束子进程占用过的CPU时间

```
clock_t times(struct tms *buf);
struct tms {
    clock_t tms_utime;  /* user  CPU time */
    clock_t tms_stime;  /* system CPU time */
    clock_t tms_cutime; /* user CPU time, terminated children */
    clock_t tms_cstime; /* system CPU time, terminated children */
};
```

## clock()

- 返回times()的四个CPU时间的总和。单位是1/CLOCKS_PER_SEC秒

## getrusage()函数，times()函数的升级版本

- 返回CPU时间，还返回表示资源使用状况的另外14个值，包括内存使用情况，I/O次数，进程切换次数

# 与时间有关的函数

- **标准函数库中time()：获得当前时间坐标**
  - 坐标0为1970年1月1日零点，单位：秒
  - t=time(0);和time(&t);都会使t值为当前时间坐标
- **函数gettimeofday()**
  - 获得当前时间坐标，坐标的0是1970年1月1日零点
  - 可以精确到微秒μs($10^{-6}$秒)
- **mktime**
  - 将年月日时分秒转换为坐标值
- **ctime()和asctime()，localtime()**
  - 坐标值和年月日时分秒转换
- **strftime**
  - 定制表示日期和时间的字符串（包括年月日时分秒等）

# 忙等待

- 多任务系统中“忙等待”的程序是不可取的

```
 1 #include <fcntl.h>
 2 main(int argc, char **argv)
 3 {
 4     unsigned char buf[1024];
 5     int len, fd;
 6     if ( (fd = open(argv[1], O_RDONLY)) < 0) {
 7         perror("Open file");
 8         exit(1);
 9     }
10     lseek(fd, 0, SEEK_END);
11     for(;;) {
12         while ( (len = read(fd, buf, sizeof buf)) > 0)
13             write(1, buf, len);
14     }
15 }
```

- 程序14行前增加sleep(1)：1秒定时轮询
- select()调用可将睡眠时间设为10毫秒级精度