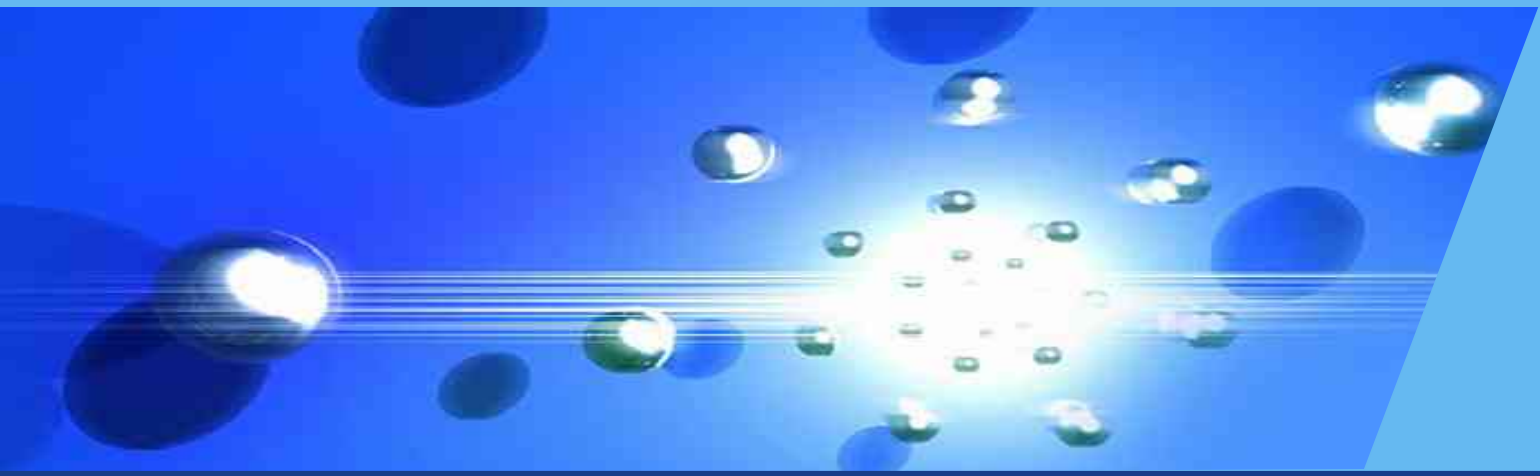
USTB

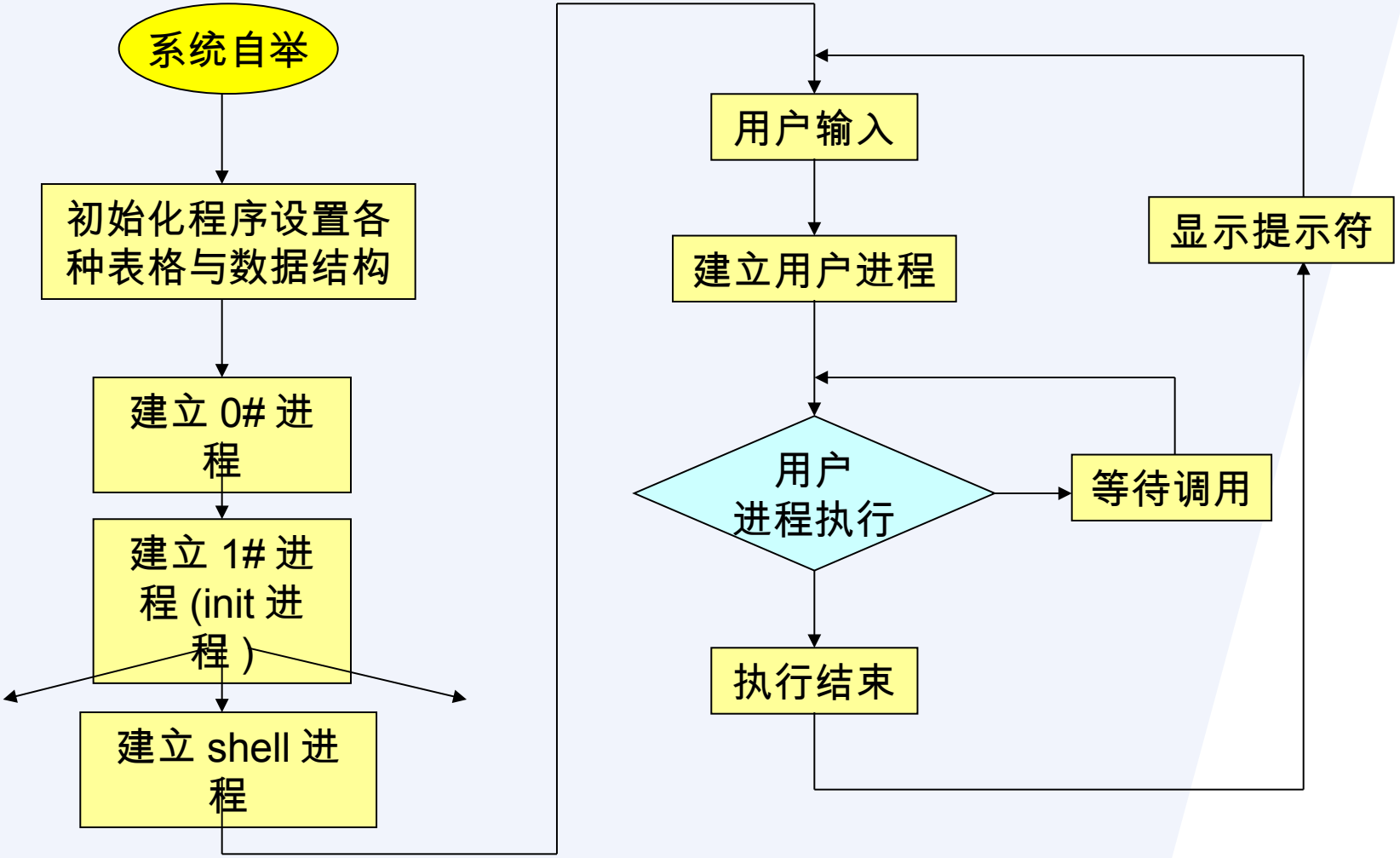


Linux 操作系统：进程与存储管理示例

本章主要内容

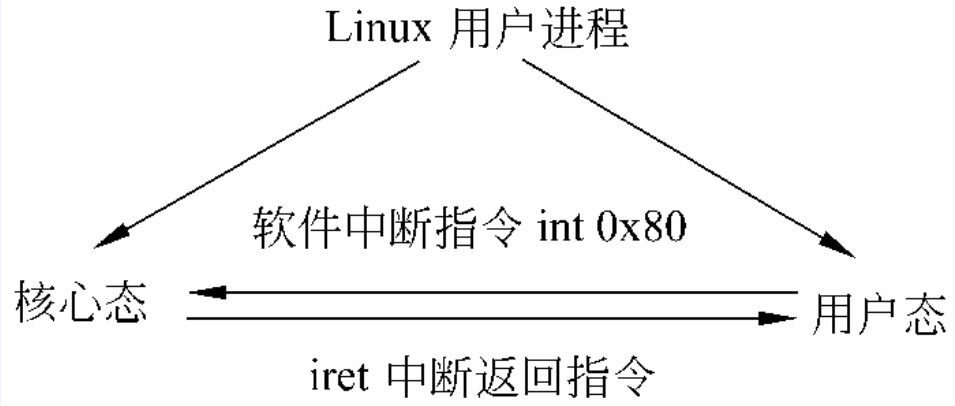
1. Linux 进程简介
2. Linux 进程结构
3. Linux 进程控制
4. Linux 进程调度
5. Linux 进程通信
6. Linux 存储管理

6.1 Linux 进程和存储管理简介



6.1 Linux 进程和存储管理简介

- **Intel 80x86 处理器的权限模式**
  - **Intel 80x86 处理器提供了四种不同权限的执行模式，但 Linux 系统只使用了 2 种：**
    - **核心态**
    - **用户态**
  - **用户进程通过 int 0x80 指令由用户态陷入核心态，且由操作系统负责切换堆栈**



6.1 Linux 进程和存储管理简介

- Linux 进程控制系统由 4 个模块组成：
  - 文件接口部分
  - 进程本身控制部分，包括进程创建、调度等
    - Linux 系统引起调度的情况：
      - 进程申请资源未得到满足，进入等待
      - 为与并发进程保持同步调用 wait 过程
      - 进程时间片用完
      - 当前进程调用 exit ，自我终止
  - 进程间控制部分，包括互斥、同步和通信等
    - 采用了 System V
  - 存储管理部分
    - 管理策略：请求调页。

6.2 Linux 进程结构

- Linux 进程的概念
  - 一个进程是对一个程序的执行
  - 一个进程的存在意味着存在一个 task_struct 结构，它包含了进程的控制信息
  - 一个进程可以生成或消灭其子进程
  - 一个进程是获得和释放各种系统资源的单位

6.2 Linux 进程结构

- Linux 进程控制块
  - task_struct 结构位于： include/linux/sched.h 用该结构来表示进程控制块，主要部分：
  - 进程状态的状态位
  - 进程标示号 pid
  - 描述进程家族关系
  - 若干用户标示号
  - 调度参数，如优先数、就绪队列、时间片等
  - 软中断信号项
  - 中断及软中断处理的有关参数
  - 各种计时项

6.2 Linux 进程结构

- Linux 进程控制块（续）
  - 进程地址空间和内存有关信息
  - 文件系统有关的信息
  - 用户文件描述符表，记录用户打开的文件
  - 进程消亡时的返回值和终止信号
  - 与进程上下文切换、现场保护有关的项
  - 资源限制有关的项

6.2 Linux 进程结构

- 进程的虚拟地址结构
  - 在 Intel 80x86 平台上，每个进程拥有 4GB 的虚拟空间
  - 0~3GB 由用户进程使用，用户进程可以直接访问
  - 3~4GB 是系统核心地址，系统进程共享。用户进程无法直接访问
  - Linux 程序由逻辑段组成，如栈段，数据段等
  - 进程的虚拟地址空间被分为若干个虚拟区域来存放相应的逻辑段。
  - 虚拟区域是进程虚拟空间上的一段连续的区域，它是被共享、保护以及进行内存分配和地址变换的实体。在 Linux 代码中通常称为 VMA

6.2 Linux 进程结构

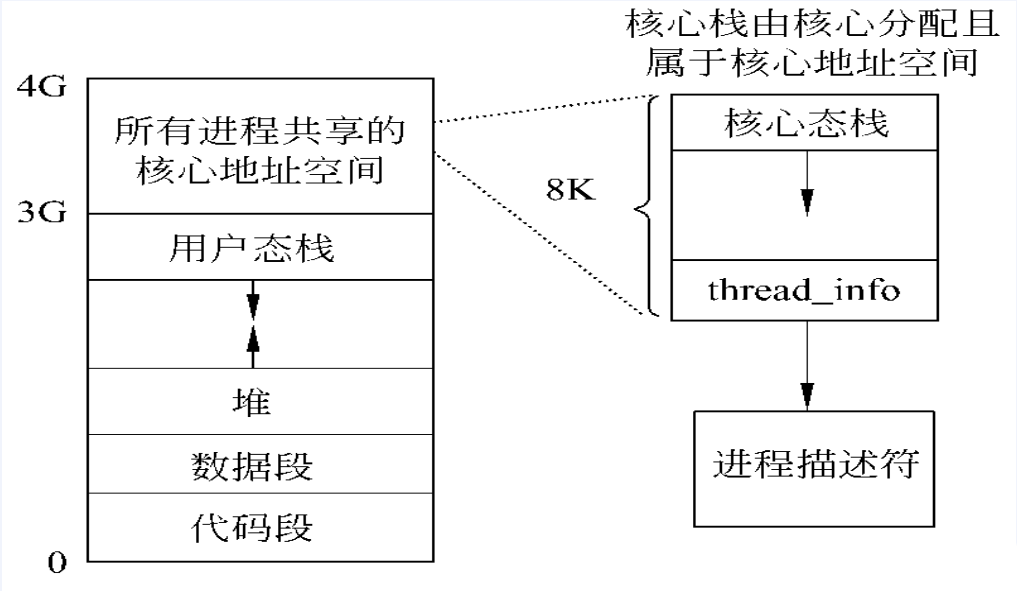
- 进程的虚拟地址结构
  - 为了管理每个进程的区，Linux 系统设置了 vm_area_struct 的数据结构，进程的每个区都有一个对应的 vm_area_struct 结构，该结构主要包括：
  - 区的标志位，指明该区的类型等
  - 区的起始地址、结束地址
  - 共享区域指针
  - 文件系统指针，执行外存中与该区对应的数据文件
  - 此区域的操作函数指针

6.2 Linux 进程结构

- 进程上下文
  - Linxu 进程上下文是由 task_struct 结构、用户栈和核心栈的内容、用户正文段、数据段、硬件寄存器的内容以及页表等组成
  - task_struct 结构
  - 寄存器包括 : 程序计数器 IP 内容 (CPU 将要执行的下条指令的虚拟地址);CPU 状态字 ; 栈指针 (指向栈中下一项的当前地址); 通用寄存器
  - 页表 (定义了进程各部分从虚拟地址到物理地址的映射)

6.2 Linux 进程结构

- 进程正文段、数据段
- 由于 Linux 采用请求页式虚存，这些段不一定总是驻留在内存空间
- 用户栈与核心栈
- 进程陷入核心程序时使用的栈，核心栈空间保存在进程的数据描述符中。

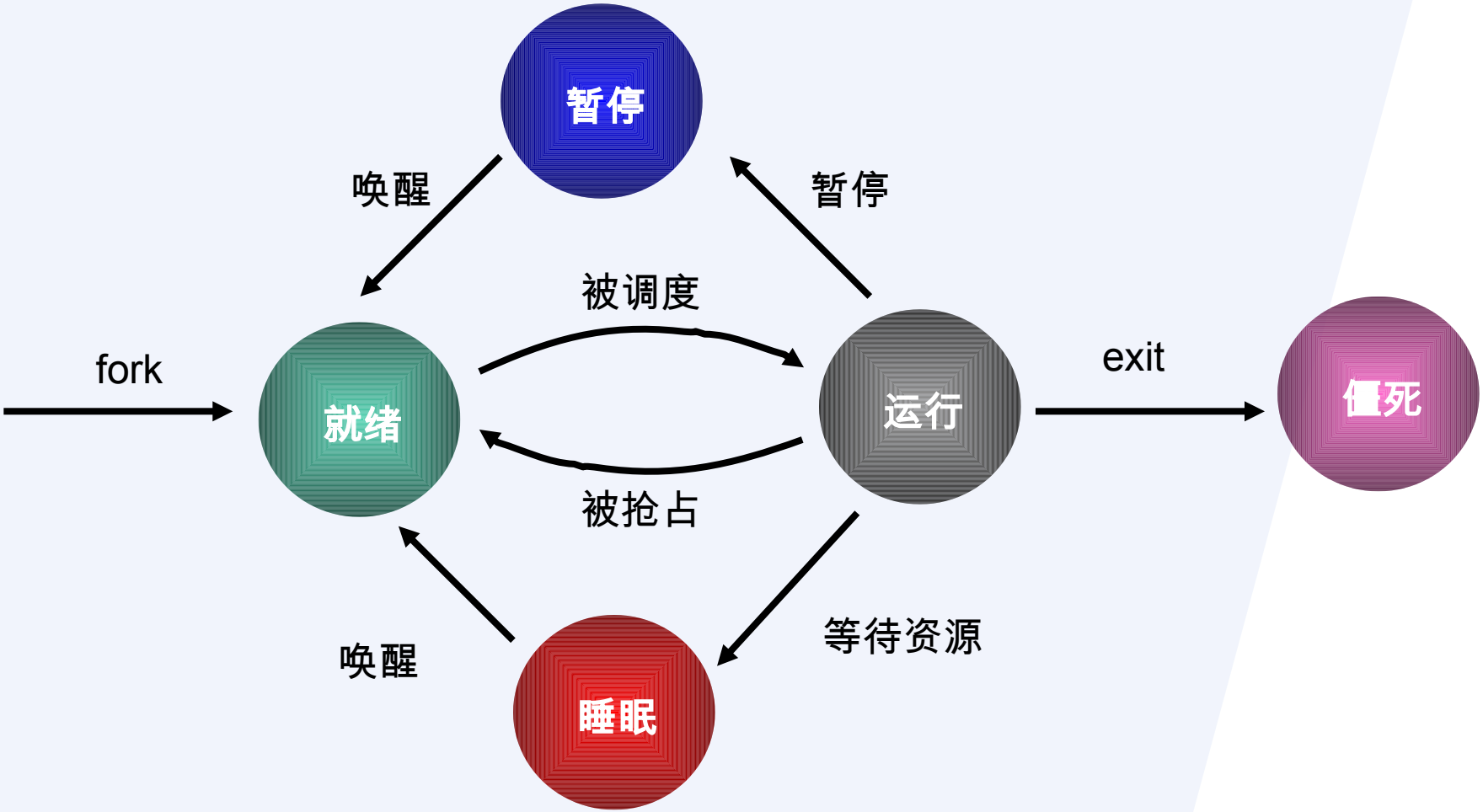


6.2 Linux 进程结构

- 进程的状态和状态转换
  - Linux 中进程共有 5 个状态
    - 1.TASK_RUNNING
    - 进程处在执行或就绪状态
    - 2.TASK_INTERRUPTIBLE
    - 进程正在睡眠，但可以被软中断信号唤醒
    - 3.TASK_UNINTERRUPTIBLE
    - 进程正在睡眠，且不可以被软中断信号唤醒
    - 4.TASK_STOPPED
    - 进程执行被暂停，当收到 SIGTOP、SIGTSTP、SIGTTIN、SIGTTOU 软中断信号后会进入这个状态
    - 5.TASK_ZOMBIE
    - 进程执行了系统调用 exit

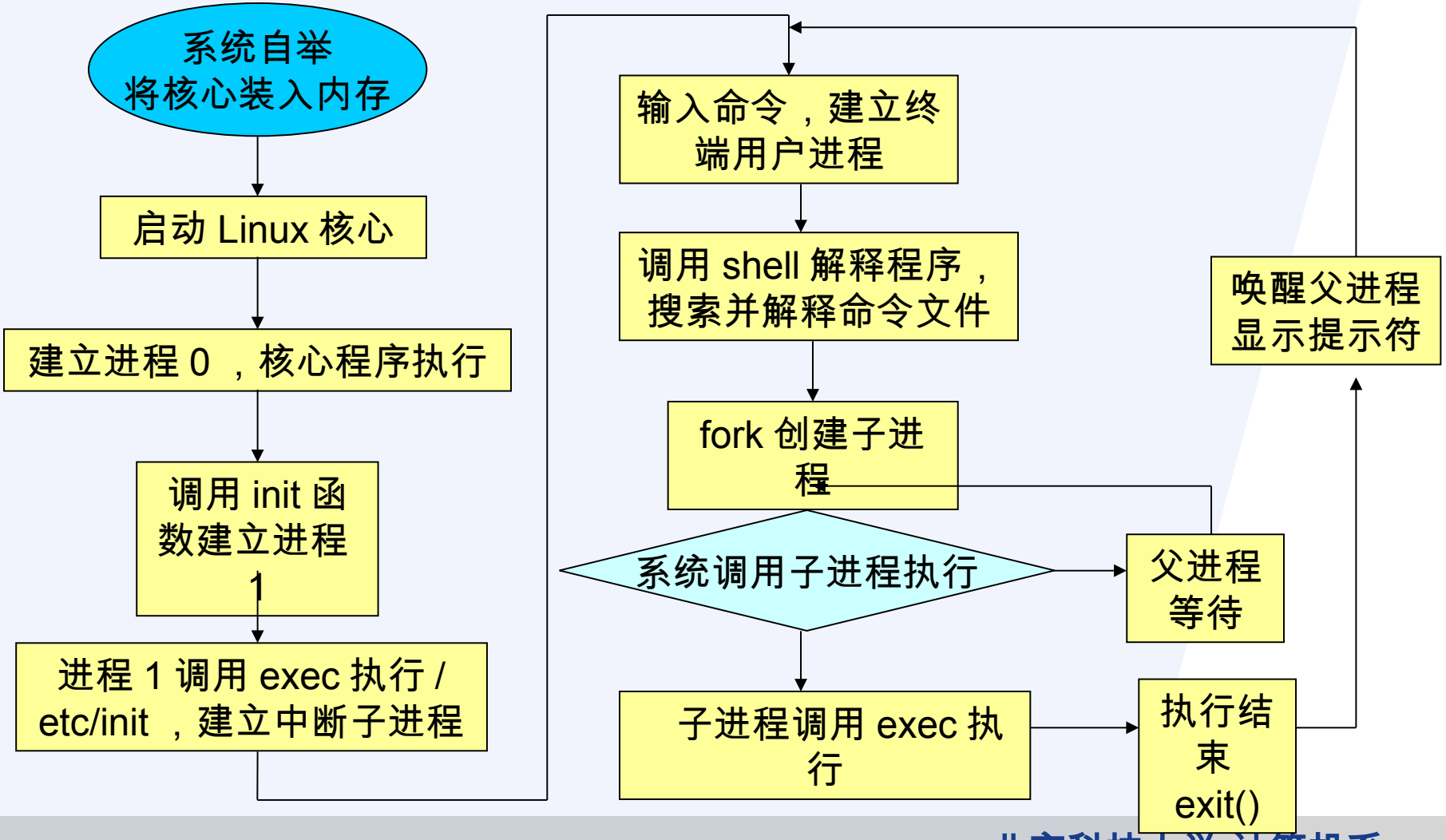
6.2 Linux 进程结构

- 进程的状态和状态转换



6.3 进程控制

Linux 启动及进程树的形成



6.3 进程控制

- 进程创建
  - Linux 系统提供 fork() 系统调用创建一个子进程，语法格式是：
  - pid=fork();
  - CPU 在父进程中， pid 为所创建子进程的进程号，若在子进程时， pid 为零
  - fork() 的源代码在 arch/i386/kernel/process.c

6.3 进程控制

- fork 过程完成的功能：
  - 为子进程分配一个进程描述符 task_struct 结构，将父进程的进程描述符内容复制到新创建的结构中，并重新设置与父进程不同的数据
  - 为子进程分配一个唯一的进程标示号 pid
  - 将父进程的地址空间的逻辑副本复制到子进程。副本为写时复制（ copy on write ）机制
  - 复制父进程相关联的有关文件系统的数据结构和用户文件描述符表，子进程就继承了父进程的文件系统相关的信息

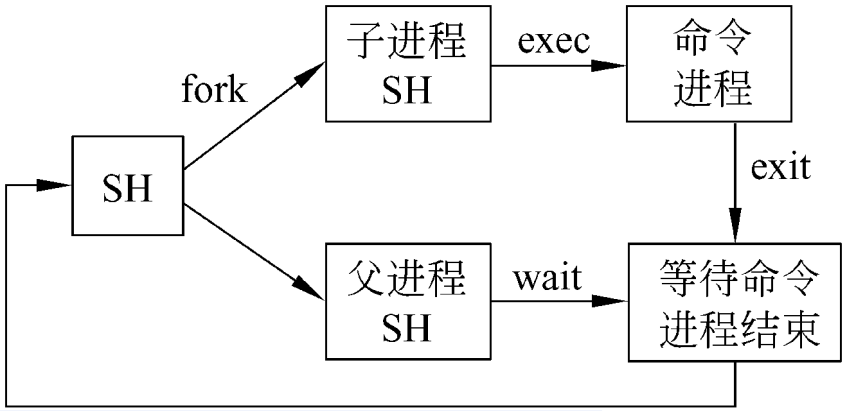
- 复制软中断信号有关的数据结构
- 设置子进程的状态为 TASK_RUNNING ，把它加入就绪队列，并启动调度程序
- 对父进程返回子进程的标示号，对子进程返回零。

执行一个文件

- 在创建一个子进程后， Linux 系统提供系统调用 exec() 来执行另一个程序。
- 系统调用 exec() 包含了六种不同的调用格式，其区别主要是在参数处理方法上，主要有：
- execvp(filename,argp)
- 或 execlp(filename,arg0,arg1 ，…， (char *)0) ；

6.3 进程控制

- 例：用 execlp 调用实现一个 shell 的基本处理过程




```
#include<stdio.h>
main()
{
        char command[32];
        char *prompt="$";
        while(printf("%s",prompt), gets(command)!=NULL){   // 打印提示符，并等待接受命令
        if(fork()==0)   // 创建子进程
                execlp(command,(char *)0);                  // 执行输入命令
        else
                wait(0);  // 父进程等待子进程结束
        }
}
```

6.3 进程控制

- 进程终止
  - 系统调用 exit(rv) 自我终止当前进程，使其进入 ZOMBIE 状态，等待父进程善后处理。
  - exit 系统调用导致释放除 task_struct 结构之外的所有资源。

6.4 Linux 进程调度

调度原理

- Linux 系统的进程调度对实时进程和普通进程采用不同的调度算法。对普通进程采用基于时间片的动态优先数调度算法。
- Linux 系统的进程调度是基于时间片加优先级，因此，进出调度主要涉及的问题：
- 调度的时机
- 调度标志设置
- 调度策略与优先数计算
- 调度的实现

6.4 Linux 进程调度

- 调度时机
  - Linux 系统发生进程调度的时机实质是：
  - 进程自动放弃处理机时，主动转入调度过程
  - 在核心态转入用户态时，系统设置了高优先级
  - 就绪进程的强迫调度标识 need_resched 时，发生调度。
- 调度标识设置
  - Linux 只使用一个调度标识 need_resched ，该标识保存在进程的进程描述符中。以下两种情况设置该标识
  - 处于运行态的进程时间片耗尽
  - 进程被唤醒，且优先级比正在运行进程的优先级高

6.4 Linux 进程调度

调度策略与优先数的计算

- Linux 把进程分为普通进程和实时进程，实时进程的调度优先级高。 Linux 系统总是优先调度实时进程，以满足实时进程对响应时间的要求。

三种调度策略

动态优先数调度：选取就绪队列中优先数最大的进程

- 优先数： weight=counter+priority-nice
- counter 是进程可用的时间片的动态优先级。
- priority 是常数，固定为 20
- nice 系统运行用户设置的一个进程优先数偏移值

- 先来先服务调度
- 轮转调度

6.4 Linux 进程调度

- 调度的实现
  - Linux 系统进程调度是由 schedule() 过程实现的。该过程源码在 kernel/sched.c 中，调度过程分为两个阶段：
  - 进行进程选择：按照调度策略，从所有进程中找到具有优先数最高的进程。
  - 进行进程切换：主要是完成进程上下文切换。


729
Prof. Lotfi Zadeh
ECSC
http://www.softcomputing.es
Evolutionary
Computational models of
processes for optimization/machi
problems that:
work over populations of potentia
the problem,
select the fittest ones for reproduct
impose different changes to them,
use a replacement mechanism to
will remain in next generations
Randomly generate initial population
Evaluate all individuals
Stop ?
yes
Best individuals
Result
no
Start
Selective reproduction
Crossing over
Mutation
Generation cycle

6.5 进程通信

Linux 的低级通信

- Linux 的低级通信主要用来传递进程间的控制信号。主要是文件锁和软中断信号机制。
- 软中断是对硬件中断的一种模拟。但是不像硬中断，只有接收进程调入执行时软中断处理程序才能生效。
- Linux 系统提供了几种相应的系统调用，如 fcntl 系统调用，利用这些系统调用可以实现锁功能。

6.5 进程通信

- 进程间通信 IPC
  - Linux 系统采用了 System V 进程间通信 IPC ， System V IPC 有 3 部分组成：
  - 消息用于进程之间传递分类的格式化数据
  - 共享存储器方式使得不同进程通过共享彼此虚拟空间而达到互相对共享区操作和数据通信的目的
  - 信号量机制用于通信进程之间的同步控制。

6.5 进程通信

- 消息机制
  - 消息机制提供了 4 个系统调用
    - int msgget(key_t key, int msgflag);
      - 创建一个新的消息队列或者访问一个已经存在的消息队列
    - int msgctl(int msqid, int cmd, struct msqid_ds *buf);
      - 对消息队列的各种操作
    - int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflag);
      - 向消息队列发送消息
    - int msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long type, int msgflag);
      - 从消息队列接收消息

6.5 进程通信

- 消息机制

顾客进程：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
struct msgform
{
        long mtype;
        char mtext[256];
}
main()
{
        struct msgform msg;
        int msgqid,pid, * pint;
        msgqid=msgget(MSGKEY, 0777);
 // 建立消息队列
        pid=getpid();
        pint=(int *)msg.mtext;
        *pint=pid;
        msg.mtpye=1;
// 指定消息类型
        msgsnd(msgqid,&msg,sizeof(int),0);
// 往 msgqid 发送消息 msg
        msgrcv(msgqid,&msg,256,pid,0);
// 接收来自服务进程的消息
        printf("client:receive from pid%d\n",*
pint);
}
```

6.5 进程通信

- 消息机制

服务者进程：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
int msgqid
struct msgform
{
        long mtype;
        char mtext[256];
} msg
main()
{
        int i,pid, * pint;
        exetern cleanup()
        for(i=0;i<20;i++)   // 软中断处理
                signal(i,cleanup);
         msgqid=msgget(MSGKEY,
0777|IPC_CREAT);
 // 建立与顾客进程相同的消息队列
        for(;;){
        msgrcv(msgqid,&msg,256,1,0);
// 接收来自顾客进程的消息
        pint=(int *)msg.mtext;
        pid=*pint;
        printf("server:receive from pid%d\n",
pid);
        msg.mtpye=pid;
        *pint=getpid();
        msgsnd(msgqid,&msg,sizeof(int),0);
// 发送应答消息
        }
```

6.5 进程通信

- 共享存储区机制
  - 进程能通过共享虚拟地址空间的若干部分，然后对存储在共享区中的数据进行读和写来直接通信。这种方式也有 4 个系统调用：
    - int shmget(key_t key, int size, int shmflag);
      - 创建新共享内存区域或返回已存在的共享区描述字。
    - void *shmat(int shmid, void *shmaddr, int shmflag);
      - 将物理共享区附接到进程虚拟地址空间。
    - int shmdt(const void *shmaddr);
      - 进程从其虚拟地址空间断接一个共享存储区。
    - int shmctl(int shmid, int cmd, shmid_ds *buf);
      - 查询及设置共享存储区状态和有关参数。

6.5 进程通信

信号量机制

- 信号量机制是基于第 3 章所述的 P 、 V 原语原理。 System V 中一个信号量由以下 4 部分组成：
  - 信号量的值，大于、小于或等于零的整数。
  - 最后操作信号量的进程的进程 id 。
  - 等待着信号量增加的进程数。
  - 等待着信号量值等于零的进程数。
- 对信号量进行创建、控制及 P 、 V 操作的系统调用：
  - semget(semkey,count,flag)
    - 创建信号量数组或查找已创建信号量数组的描述字。
  - semop(semid,oplist,count)
    - 用于控制 P 、 V 操作。
  - semctl(semid,number,cmd,arg)
    - 对信号量进行控制操作的系统调用。

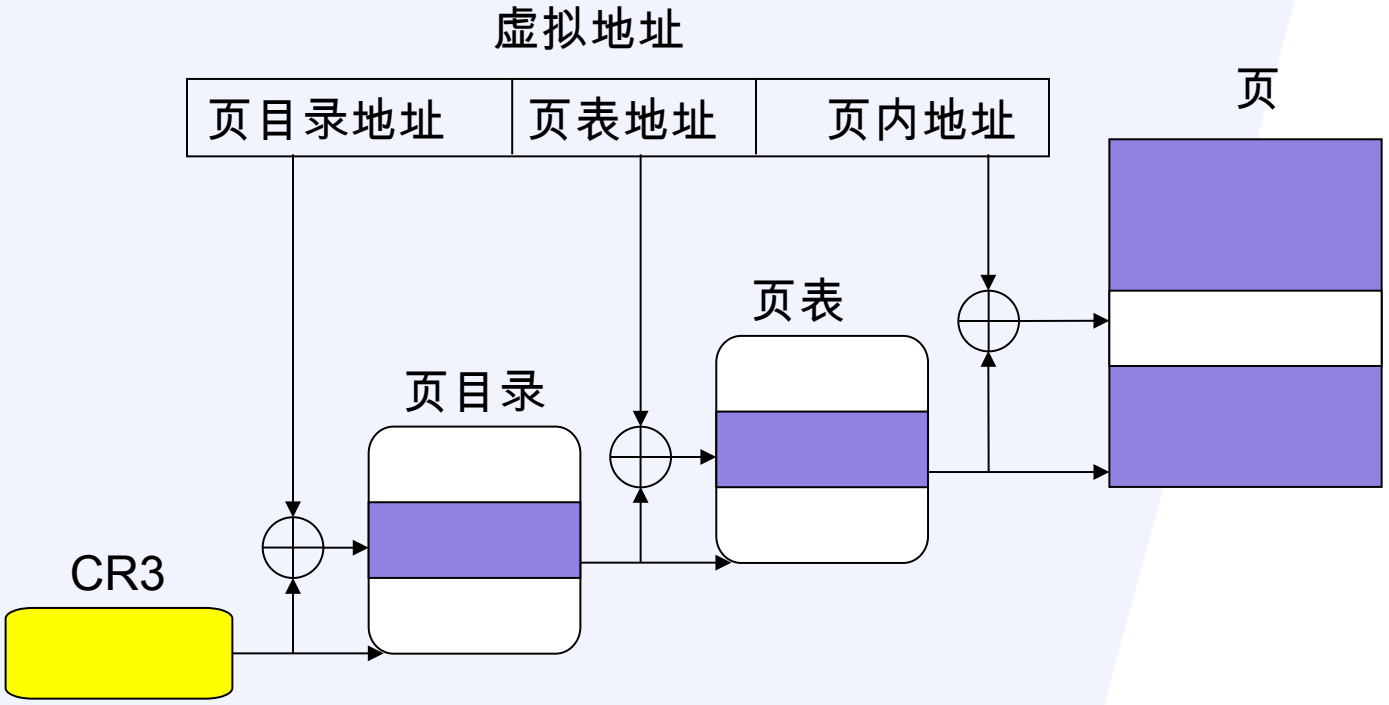
6.6 Linux 存储管理

- 虚拟空间和管理
  - 内存管理与硬件体系结构相关，所以先要了解 Intel 80x86 处理器：
    - Intel 80x86 处理器支持段页式的内存管理。 其中分段单元由 6 个段寄存器 .
    - 逻辑地址由段标识符与段内地址组成
      - 如 cs:0x1000 表示逻辑地址是 cs 段，段内偏移为 0x1000
    - 段的详细地址由 8 字节的段描述符表示。包括起始地址，长度，段的特征等。
    - 80x86 的分页单元由页目录基地址寄存器 CR3 以及页目录表、页表项共同作用，用于将分段单元转换后的虚拟地址转换为相应的物理地址。

6.6 Linux 存储管理

虚拟空间和管理

- 但 Linux 系统实际上非常有限的使用了 80x86 的分段机制。可以忽略其分段机制的处理，主要采用页式管理



6.6 Linux 存储管理

请求页技术

- 当请求页不在内存中时，会产生缺页中断，如果不是非法访问，将进入缺页处理过程。其原因有 3 种：
  - 该页首次访问，还没有分配物理空间
  - 在外存的文件中，如可执行文件
  - 在外存的交换区中

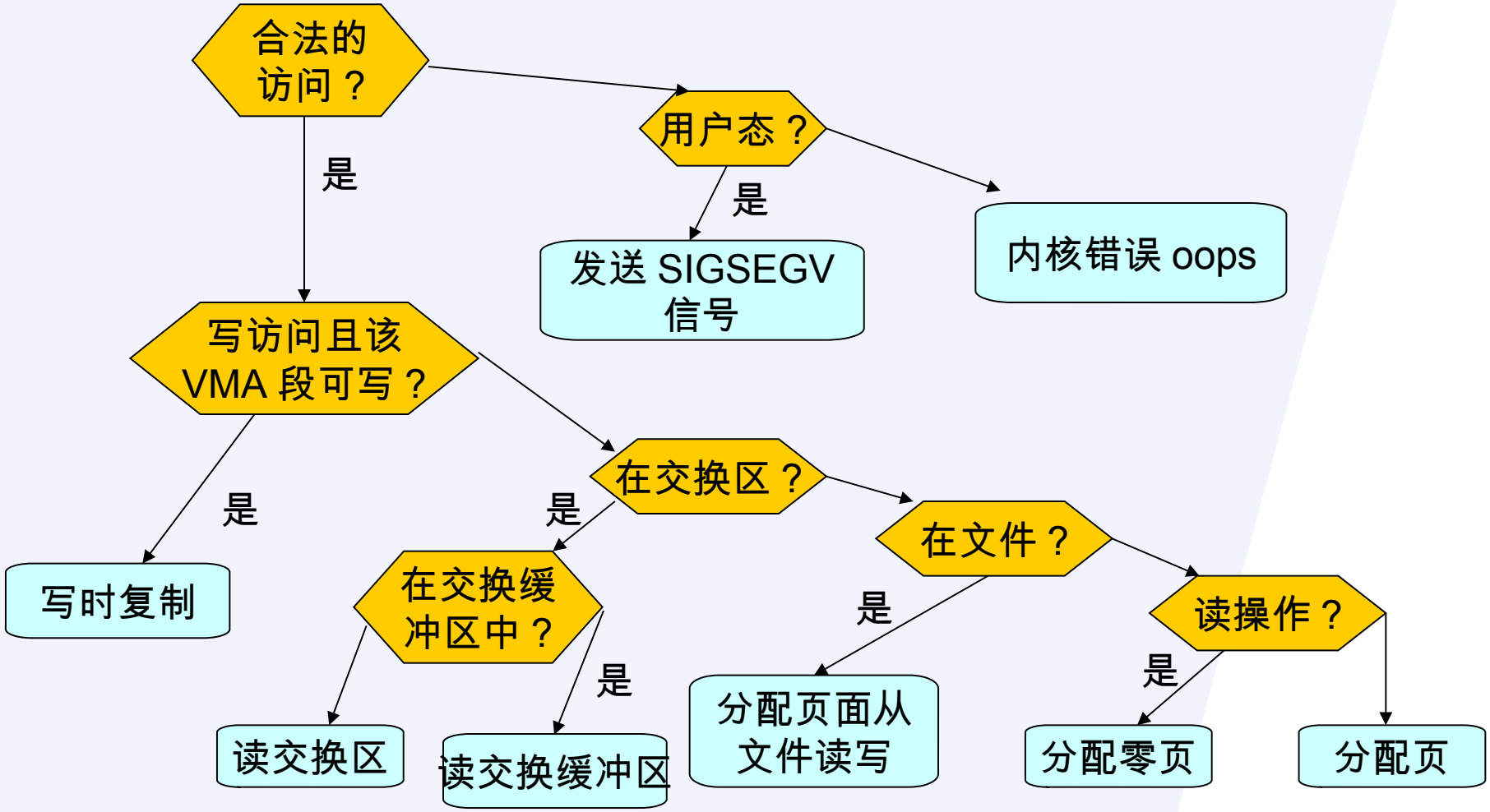
6.6 Linux 存储管理

写时复制机制

- 是一种可以推迟甚至避免拷贝数据的技术。
- 当子进程被创建时，内核此时并不复制整个父进程的地址空间，而是让父子进程共享同一个地址空间。只用在需要写入的时候才会复制地址空间，从而使各个进程拥有各自的地址空间。
- 即资源的复制是在需要写入的时候才会进行，在此之前，只以只读方式共享。这种技术使地址空间上的页的拷贝被推迟到实际发生写入的时候。

6.6 Linux 存储管理

- 页面调入 (交换缓冲条件下)



6.6 Linux 存储管理

页面换出过程

- Linux 在两种情况下进行页面换出工作
  - 分配内存的时候发现空闲内存低于某一个极限值
  - kswapd 核心线程每 10 秒一次周期性的换出内存
- 换出操作入口 try_to_free_pages() 函数，源码在 mm/vmscan.c 中，主要过程：
  - 检查缓冲区，释放 inactive_list 中没被对象引用的页
  - 如果无法满足，则遍历各进程的地址空间，检查页面对应的页表项的访问标志，不为真者视为最近没访问，将其释放。

小结

- Linux 进程结构
- Linux 进程的创建
- Linux 进程间的通信机制
- Linux 内存管理


2005. 11. 14