

# 《操作系统设计与实现》

## 操作系统的实现

主讲教师：史广顺

[gsshi@expervision.com.cn](mailto:gsshi@expervision.com.cn)

[gsshi@imi.nankai.edu.cn](mailto:gsshi@imi.nankai.edu.cn)

# 内容提要

- ❖ 代码结构组织与数据结构定义
- ❖ 专题1——操作系统的启动过程
- ❖ 专题2——操作系统的进程管理
- ❖ 专题3——操作系统的设备管理
- ❖ 专题4——操作系统的内存管理
- ❖ 专题5——操作系统的文件管理

# 头文件的组织方法

## ❖ 分类保存头文件

- **include/**: 最基本的公共头文件（不依赖软硬件环境）
- **include/sys**: 遵循系统规范的头文件（**Posix**规范）
- **include/minix**: 实现系统内部功能的头文件
- **include/ibm**: 针对特定硬件平台的头文件

## ❖ 头文件分类组织思想的讨论与反思

- 分目录保存头文件，保持系统结构清晰，便于引用
- 分类别定义头文件，便于代码维护和扩展
- 对复杂系统而言，头文件的组织和定义是第一个难题

# 头文件中的玄机探密

## ❖ 预编译选项的应用

- 实现自定义的系统配置（分析与举例）
- 与编译器相互结合，实现外部环境侦测（分析与举例）
- 实现功能的扩展与限制（分析与举例，**termios.h**）
- 组合形成的复杂应用（**Minix**中的**Error code**定义举例）

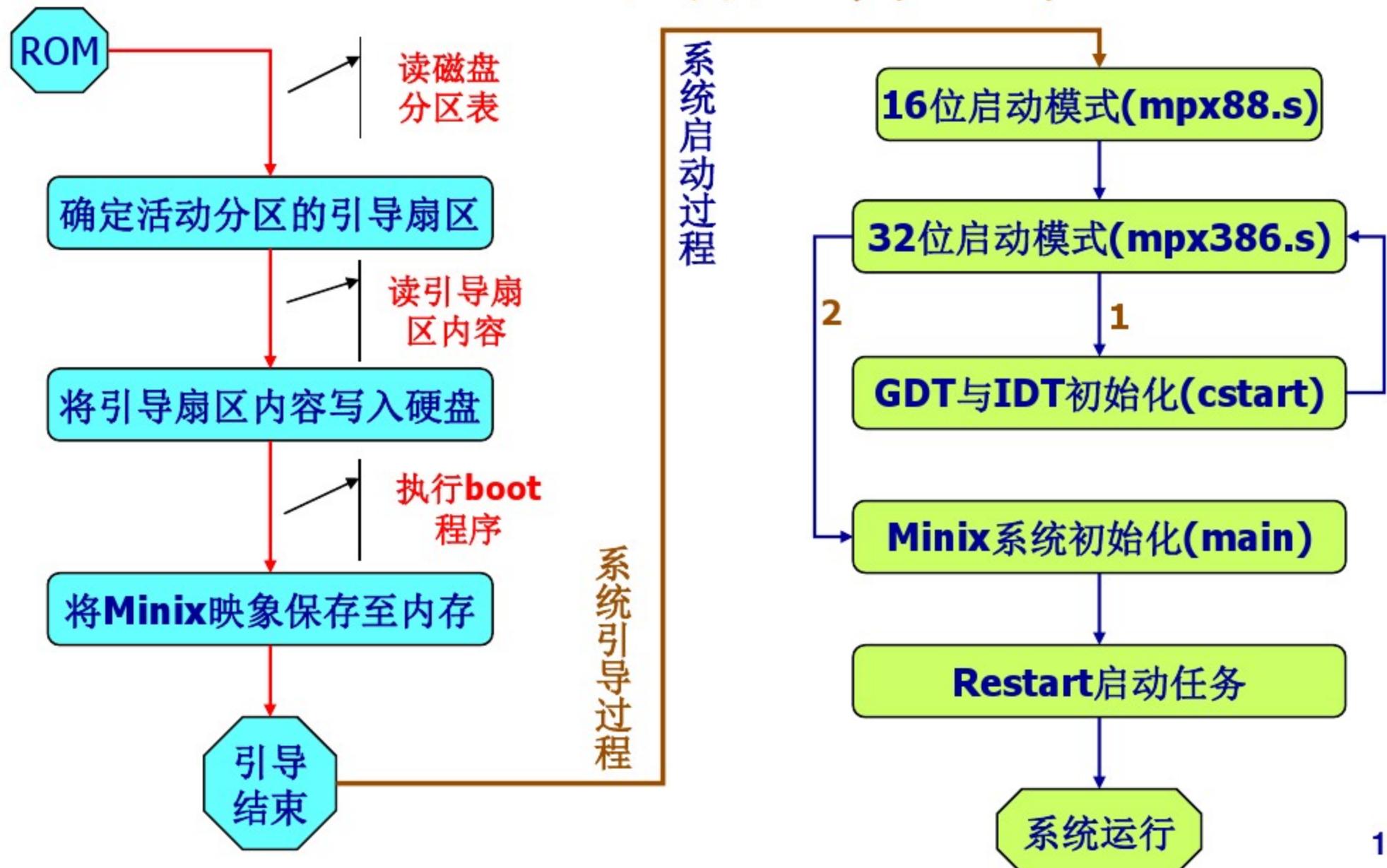
## ❖ 头文件的实现规范

- 防止重复引用的错误（分析与举例，**const.h**分析）
- 支持不同语言的结合（分析与举例）
- 头文件的包含次序（分析与举例）
- 全局变量的定义与实现（**Table.c**分析）

# 内容提要

- ❖ 代码结构组织与数据结构定义
- ❖ 专题**1**——操作系统的启动过程
- ❖ 专题**2**——操作系统的进程管理
- ❖ 专题**3**——操作系统的设备管理
- ❖ 专题**4**——操作系统的内存管理
- ❖ 专题**5**——操作系统的文件管理

# Minix的启动过程



# Minix的系统引导过程

## ❖ 硬件启动的引导过程

- **BIOS**规范设定了对磁盘分区的要求，第一扇区为引导扇区
- 操作系统遵循**BIOS**规范，在引导扇区内保存系统引导程序
- **BIOS**程序自动寻找磁盘引导扇区，执行系统引导程序
- 系统引导程序读取引导选项，开始正常启动操作系统

## ❖ 引导过程分析

- 活跃分区、引导扇区、磁盘分区表的组织和使用
  - 操作系统的安装过程：**Minix**中的**Installboot**程序
  - 操作系统的启动过程：**Minix**中的**Boot**程序
- 引导过程的参数设置与管理（举例：多系统引导是如何实现的？）
  - **Minix**：引导块为**1K**，第二扇区保存引导选项
  - **Windows**：**Ntldr**寻找**Boot.ini**（遵循**ARC**规范），再确定引导内容

# 磁盘分区的组织与管理

## ❖ 未分区的磁盘

- 第一扇区为引导扇区
- 由引导程序直接载入即可

## ❖ 已分区磁盘

- 必须存在主分区，主分区的第一扇区保存**MBR**
- **MBR**中包含分区表，各个分区第一扇区为引导扇区
- 从主分区一直读取到系统所在分区的引导扇区
- **Windows**中的引导卷和系统卷区别

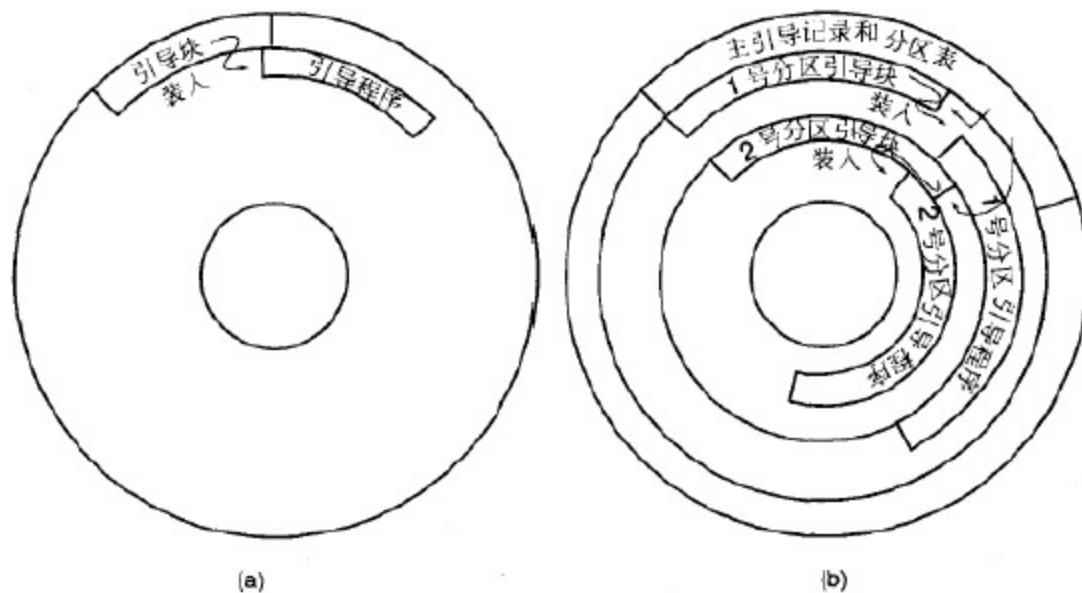


图 2-31 引导使用的磁盘结构。(a)未分区的磁盘, 第一个扇区就是引导块。  
(b)分过区的磁盘, 第一个扇区是主引导记录

# Minix的系统初始化过程

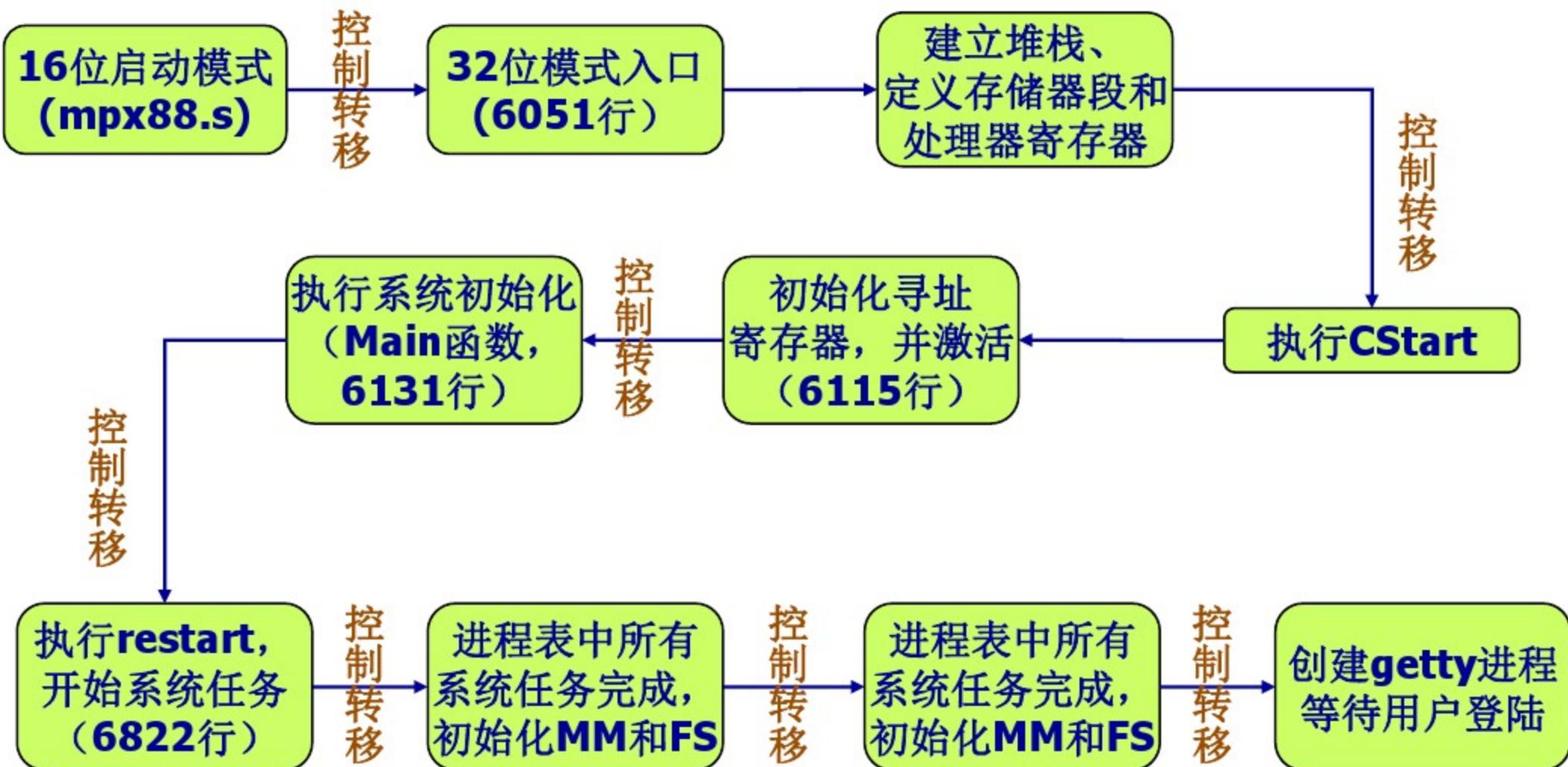
## ❖ 操作系统与CPU的兼容

- **16位/32位模式是CPU**（微处理）的硬件规范
- 操作系统必须与**CPU**的硬件规范兼容，因此针对**16位/32位**存在不同的初始化过程，执行初始化过程的程序被称为引导监控程序
- 初始化过程基本都是使用汇编语言，在适当的时候控制权转移高层程序
- **Minix**中的初始化代码：**mpx.s(5800行)**
- 初始化过程都是首先按照**16位**模式运行，在适当时候切换至**32位**模式

## ❖ Minix的系统初始化过程

- **16位模式运行**——**切换至32位(6051行)**——建立堆栈框架运行**C程序**——**定义存储器段**（拷贝**CPU内表格**）——**CStart**（初始化**GDT**和**IDT**）——**填充寻址寄存器的初值(6115行)**——**激活GDT和IDT**——**执行高层系统初始化(6131行)**——**Restart**启动系统任务——**系统任务阻塞**——**MM和FS**初始化——**getty**进程——用户登陆
- 初始化过程的控制流程
  - **mpx88.s**——**mpx386.s**（汇编——**CStart**——汇编——**Main**——汇编——**MM/FS**——用户）

# Minix的启动过程（Init进程）



# Minix系统初始化（Main）



## ❖ 了解**Main**函数需要哪些知识

- 进程表、内存管理的数据结构
- 中断控制管理的结构与机制
- 系统任务的类型、内容
- 进程表初始化的细节操作
- 系统任务启动的处理过程

## ❖ 如何获取以上相关知识

- 阅读头文件了解各类数据结构
- 阅读被调用函数代码了解处理机制
- 阅读处理代码了解初始化细节操作
- 总结分析，理解系统初始化过程

# 进程、内存、系统任务结构定义

## ❖ Minix中的进程结构

- **kernel\Proc.h** 中定义，注意各个数据成员的含义。**Proc**与**Proc[]**结构的重复定义

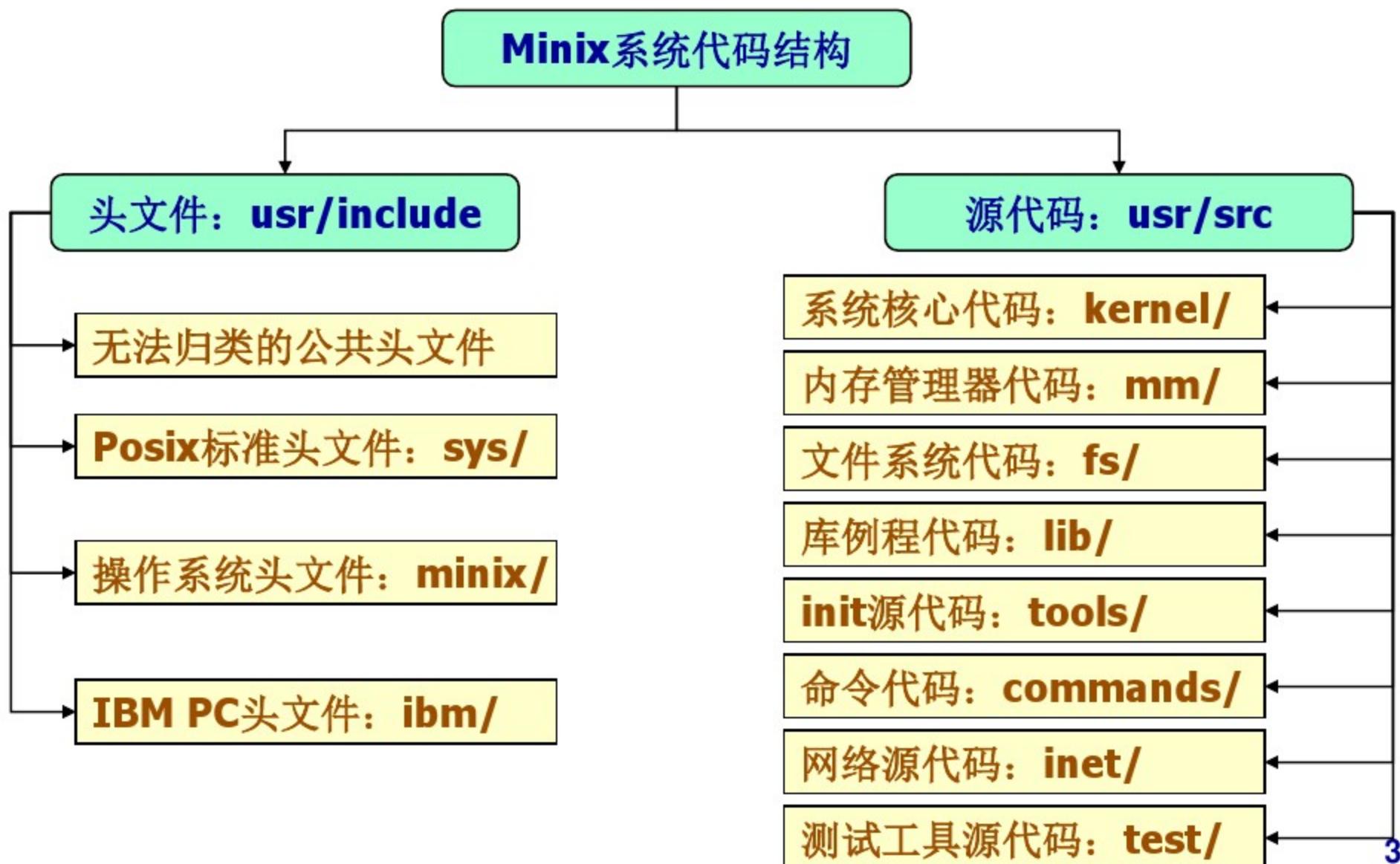
## ❖ Minix中的内存结构

- **minix\type.h** 中定义，**mem\_map**结构

## ❖ Minix中的系统任务结构

- **kernel\table.c** 中定义，注意**Extern**的用法

# 代码结构组织



# 内容提要

- ❖ 代码结构组织与数据结构定义
- ❖ 专题**1**——操作系统的启动过程
- ❖ 专题**2**——操作系统的进程管理
- ❖ 专题**3**——操作系统的设备管理
- ❖ 专题**4**——操作系统的内存管理
- ❖ 专题**5**——操作系统的文件管理

# 进程管理的基本原理回顾

## ❖ 进程的基本概念

- 进程与程序的差别、进程状态

## ❖ 进程调度基本原理

- 调度策略的评价标准：公平、有效、响应时间、周转时间、吞吐量
- 调度模型：时间片轮转、优先级、多重队列、最短作业优先、彩票调度、实时调度、两级调度
- 进程调度的设计思考：调度策略与调度机制分开

## ❖ 进程通信基本原理

- 进程通信的三面内涵：信息传送、互斥、同步
- 进程通信基本概念：竞争条件、临界区、互斥解决方案的四个条件
- 进程互斥的解决方法：忙等待、睡眠/唤醒、信号量、消息机制、管程
- 进程同步的解决方法：信号量、消息机制

# Minix中的进程调度

## ❖ 基本数据结构

- 三级进程队列: **rdy\_head[NQ]**、**rdy\_tail[NQ]** (**kernel\proc.h**)
- **proc\_ptr**: 全局变量, 指向当前应该运行的进程

## ❖ 调度相关函数 (**kernel\proc.c**)

- 选择合适的进程运行: **pick\_proc**函数 (7176行)
- 进程状态转化: 变为就绪态和被阻塞, **ready**函数和**unready**函数 (7207行)
- 用户进程的时间片轮转: **sched**函数 (7308行)

## ❖ 关于进程调度机制实现过程的思考

- 如何启动进程调度机制呢?
- 如何实现进程状态的自动转化呢?
- 如何保证OS运行的稳定呢? (关中断的锁变量)

# Minix中的进程通信

## ❖ 进程通信的基本知识

- 操作系统对中断（硬件中断和软件中断）的响应机制
- 消息传递的运行机制（还记得消息机制的实现方法吗）

## ❖ Minix中断处理机制

- 对**8259**芯片的操作（注意保护模式与实模式的差别）
- 对中断号、中断向量、中断描述表的操作
- 对硬件中断、软件中断（系统调用）的操作

## ❖ Minix进程通信机制

- 对中断的响应（将中断转化为消息进行处理）
- 进程间通信的处理（系统任务和服务器进程、用户进程与其他进程）
- 通信机制的实现（控制流程）

# Minix中的进程通信

## ❖ Minix进程通信原理

- 基于会合机制的消息传递，**Senc**与**Recv**的同步
- 核心函数：**mini\_send**和**mini\_rec**函数（7045行，仔细阅读）
- 最困难的理解：如何完成硬中断和软中断的消息传递

## ❖ Minix硬件中断处理机制

- 8259芯片工作原理（图2-33）
- 硬件中断信号——8259响应——中断信号处理——硬件底层历程——**Interrupt**调用——系统任务
- 结合8259中断控制器、CPU保护模式、进程消息通信机制的复杂过程

## ❖ Minix软件中断处理机制

- 系统调用函数——构造消息——int **SYS386\_VECTOR**指令——\_s\_call调用——**sys\_call**调用——系统任务
- 结合库函数、进程消息通信、CPU保护模式的复杂过程

## ❖ 对硬件中断和软件中断的理解和学习

- OS中进程通信的最底层操作就是对硬件中断和软件中断的响应与处理。
- 复杂的中断嵌套情况真实体现了多进程并发的核心本质。

# Minix中的进程通信

- ❖ 中断处理硬件的基本工作原理
  - 主从中断控制器
  - 中断捕获、发送、处理基本流程
- ❖ 中断门描述符（8字节结构）
  - 说明了处理该中断所需的核心例程
  - 中断处理软件的入口
- ❖ 中断处理的基本过程
  - CStart中建立中断表
  - Main中对中断控制器进行初始化
  - 捕获中断后执行相关历程

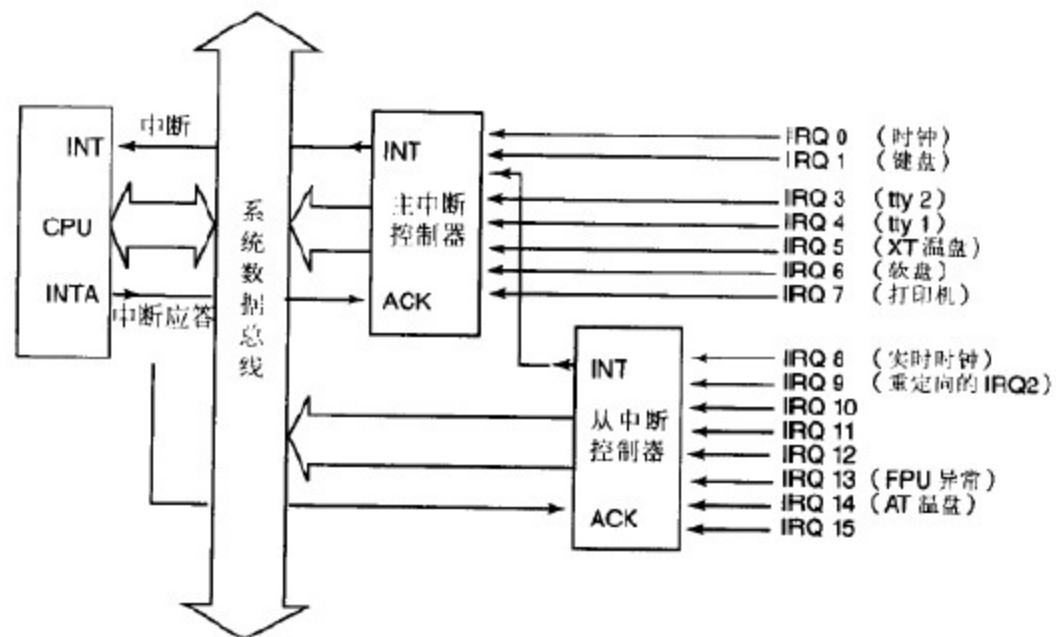


图 2-33 一台 32 位 Intel PC 上的中断处理硬件

# Minix中的中断处理

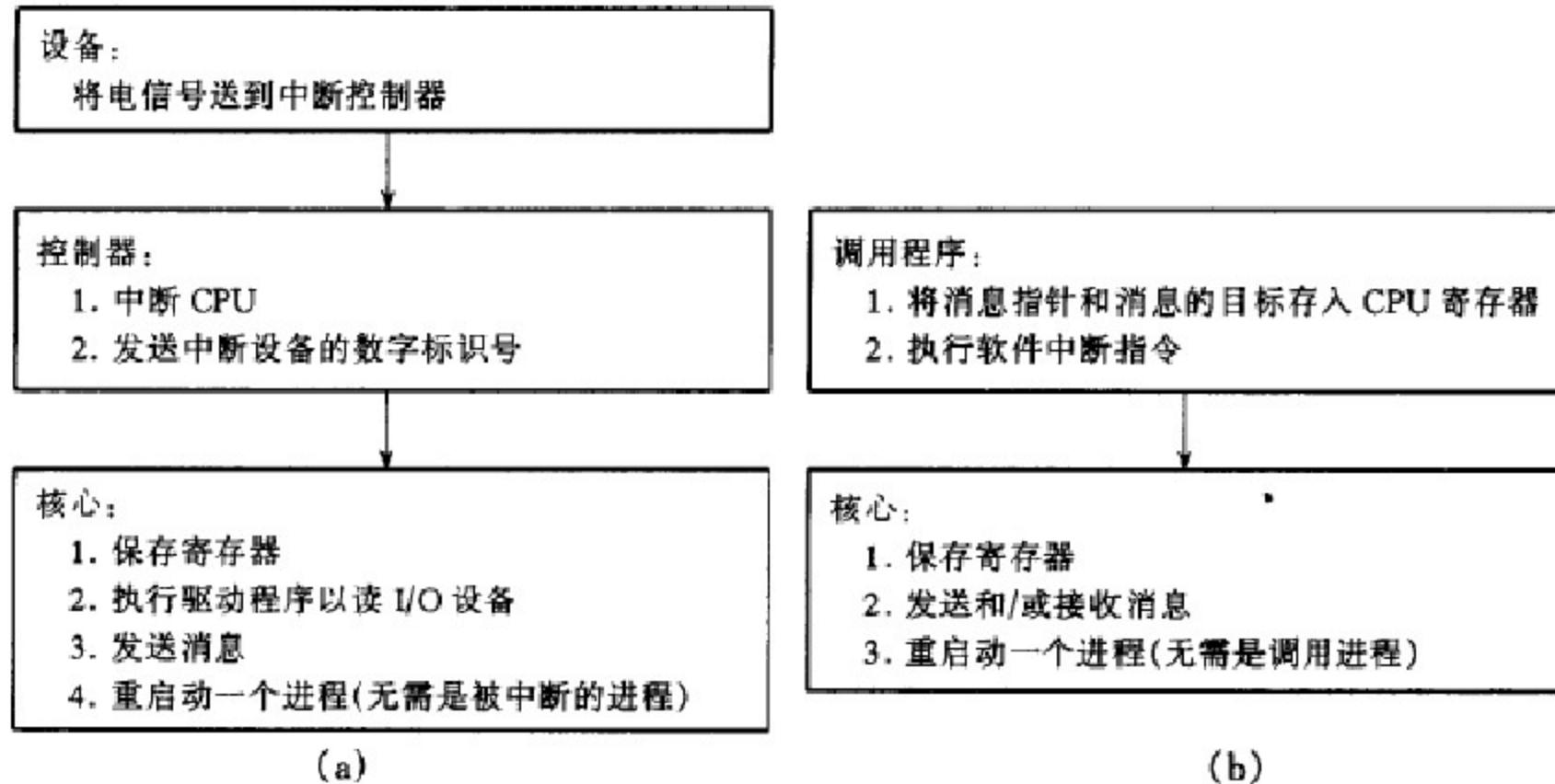
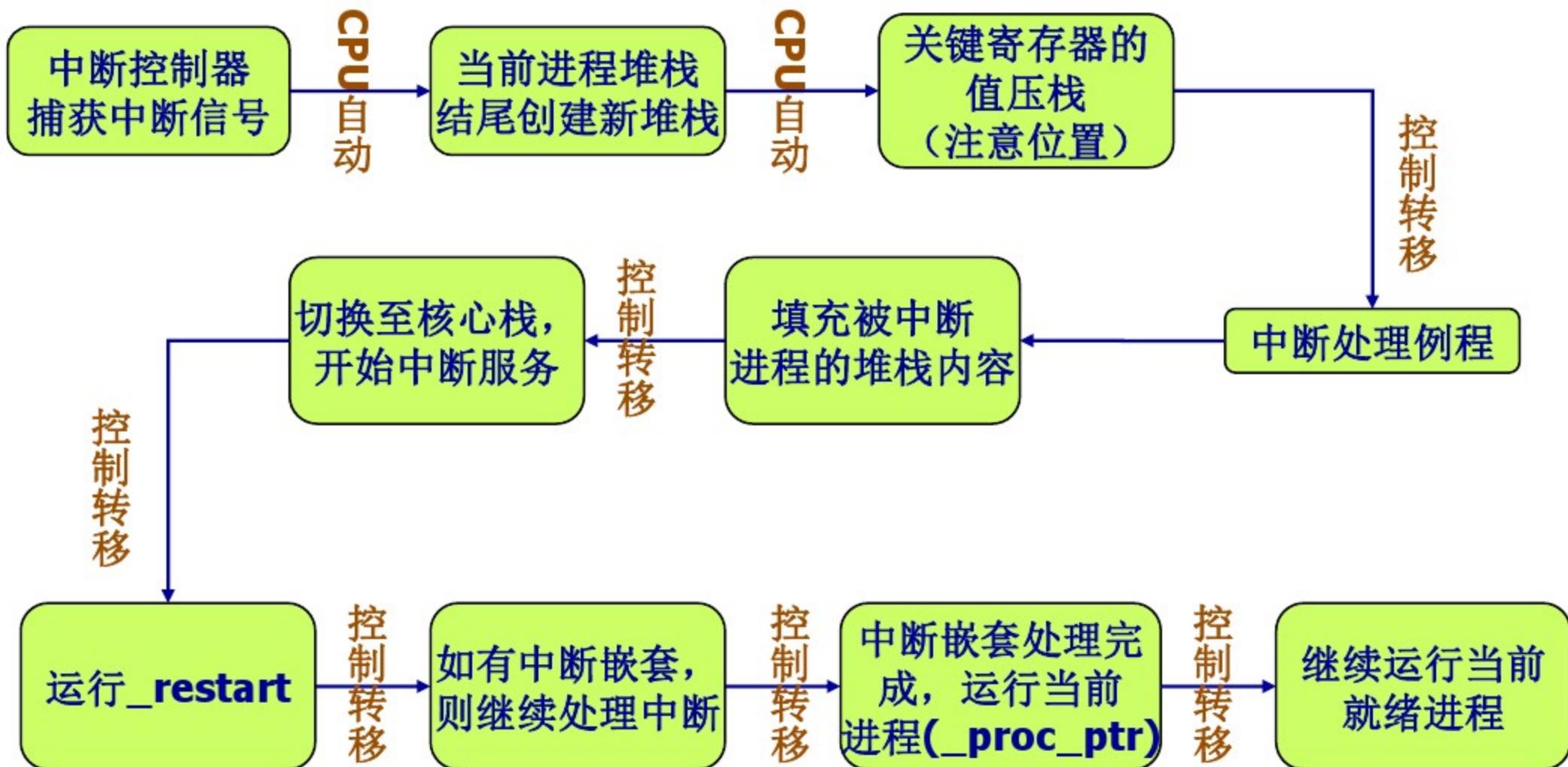


图 2-34 (a)硬件中断的处理过程。(b)系统调用处理过程

# Minix中的中断处理



# Minix中的中断处理

## ❖ 中断处理所使用的数据空间

- 进程堆栈、**TSS**（任务状态段，核心栈）
- 中断处理例程使用进程堆栈（保存上下文、记录返回地址和信息）
- 中断服务例程将使用核心栈（完成中断服务，调用**Interrupt**，引起进程调度）

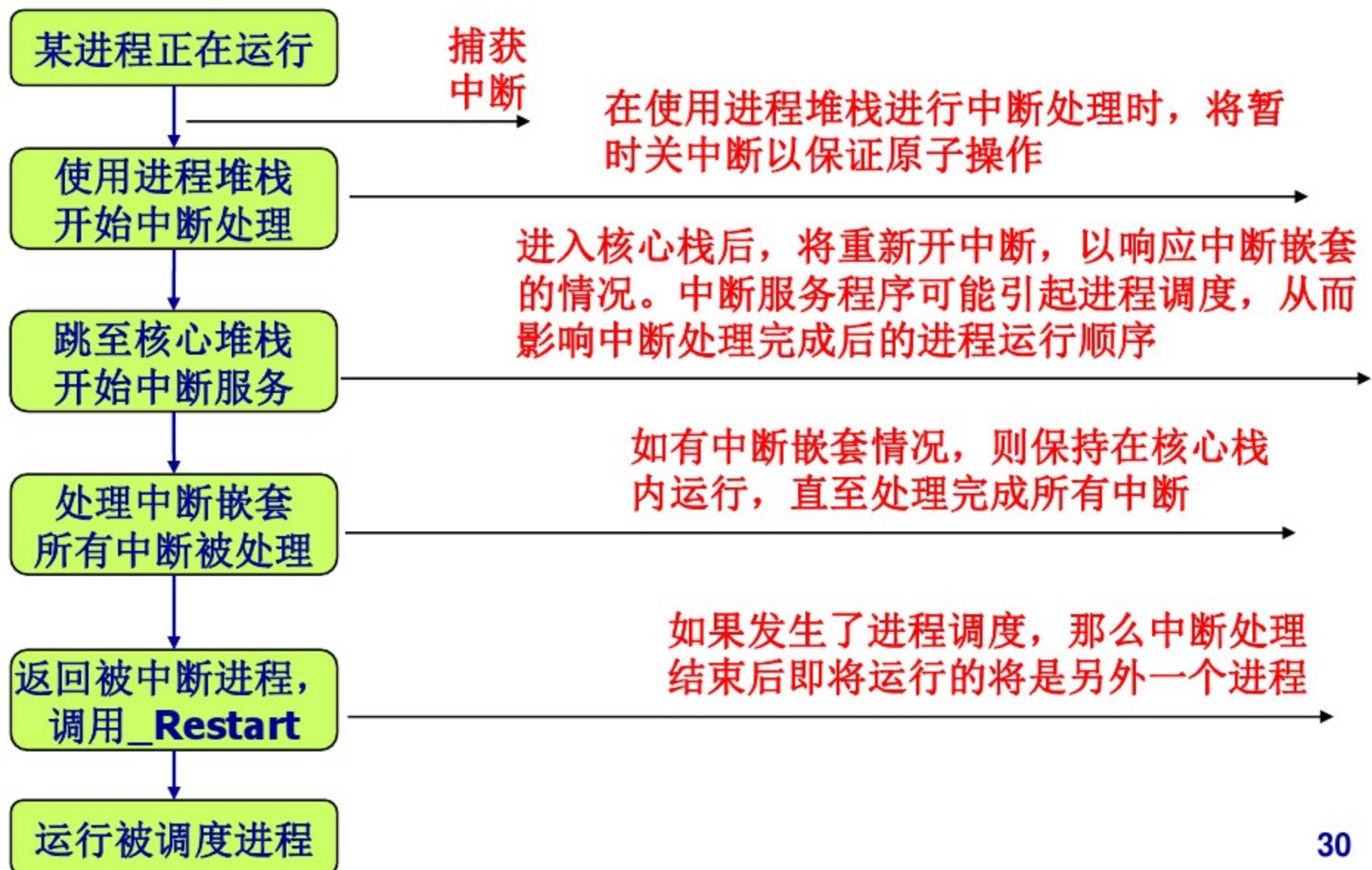
## ❖ 中断结束时发生的情况

- 在中断服务期间，可能引起进程调度，改变**\_proc\_ptr**的值
- 中断服务结束后，将首先检查是否存在中断嵌套情况（**\_k\_reenter**）标识
- 处理完所有中断后，将从核心栈跳回至进程堆栈，然后调用**\_restart**

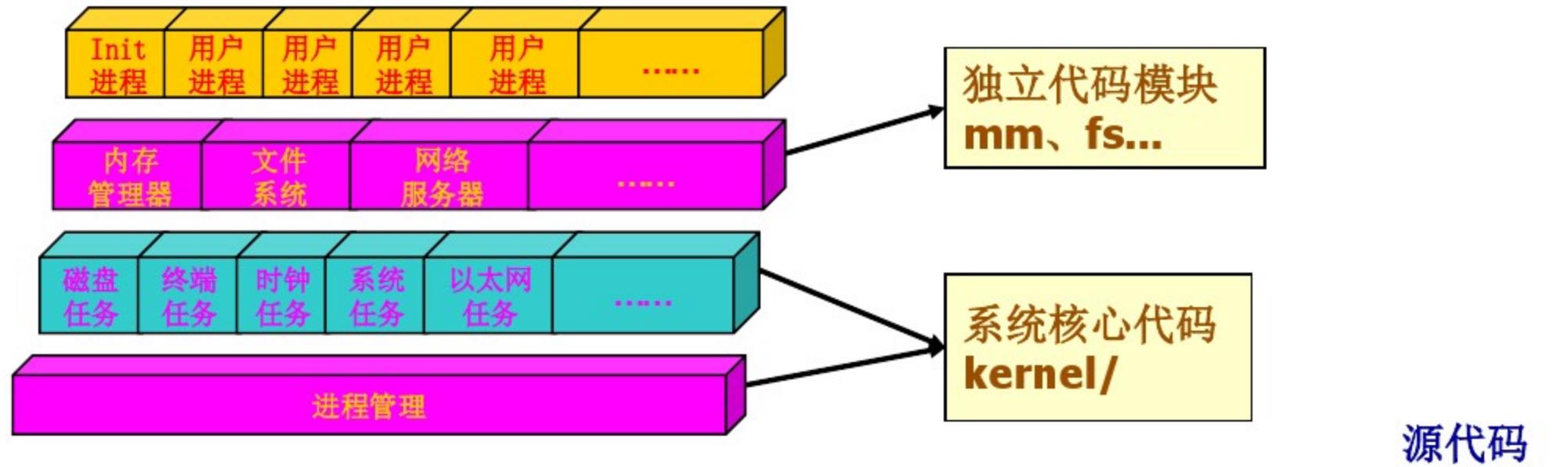
## ❖ 多进程并发的理解

- 中断服务期间，可能引起进程调度，改变**\_proc\_ptr**的值
- 中断服务结束后，**\_restart**将选择**\_proc\_ptr**作为下一个运行的进程
- 被中断的进程有可能继续存在于就绪队列中，但未必会立刻运行

# Minix多进程并发的核心思想

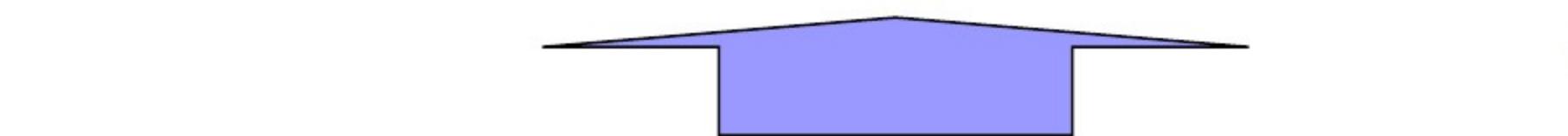


# 代码结构组织

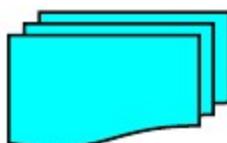
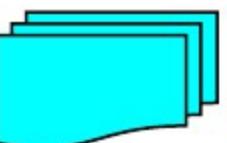
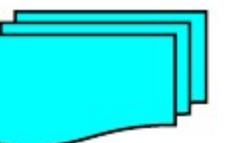
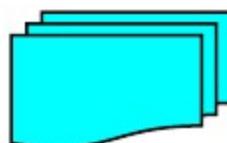
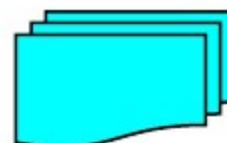


源代码

头文件



系统配置定义

编译环境  
定义硬件环境  
定义数据类型  
定义DS、接  
口定义常量与  
宏定义全局变  
量定义

# 内容提要

- ❖ 代码结构组织与数据结构定义
- ❖ 专题**1**——操作系统的启动过程
- ❖ 专题**2**——操作系统的进程管理
- ❖ 专题**3**——操作系统的设备管理
- ❖ 专题**4**——操作系统的内存管理
- ❖ 专题**5**——操作系统的文件管理

# Minix中的时钟设备管理

## ❖ 时钟设备的服务目标

- 维护日期时间
- 基于时间片轮转对用户进程进行调度
- 对CPU的使用进行记帐
- 定时器服务——闹钟（用户进程）、监视定时（系统任务）
- 直方图检视和统计信息搜集

## ❖ Minix时钟任务的设计原理

- 两个系统任务：**Clock\_Task**和**Syn\_Alarm\_Task**
- 所涉及的各种方法：时间统计、定时器处理、时间片调度
- 所使用的机制：基于消息的进程通信、硬件中断和软件中断处理

## ❖ 时钟中断的处理

- 硬件中断：**8259**捕获——底层驱动历程——发送消息——系统任务
- 软件中断：**CPU**指令——**sys\_call**调用——发送消息——系统任务

## ❖ 对时钟设备管理的原理掌握和实现过程理解

- 具备完整的中断、消息知识体系
- 针对各种目标，了解处理过程和实现机制

# Minix时钟设备基本原理

## ❖ 时钟硬件设置

- 可编程时钟，方波模式（**60HZ**）
- 时间片设定：**100**微秒

## ❖ 时钟服务实现方法

- 系统时间维护：启动时间+消耗滴答
- 定时器实现：虚拟时钟链表

## ❖ 时钟服务的重要函数

- 硬件中断处理：**clock\_handler**
- 软件中断处理：**do\_\*\*\*\***和**syn\_alarm\_task**

## ❖ 函数层次的理解和掌握

- 最底层：**clock\_handler**（硬件中断）
- 中间层：**clock\_task**和**syn\_alarm\_task**（系统任务）
- 最高层：**do\_\*\*\*\***和其他功能函数（实现代码）

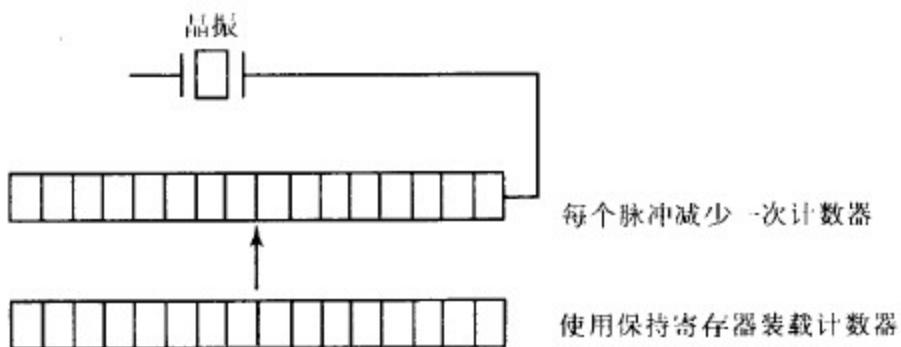


图 3-23 可编程时钟

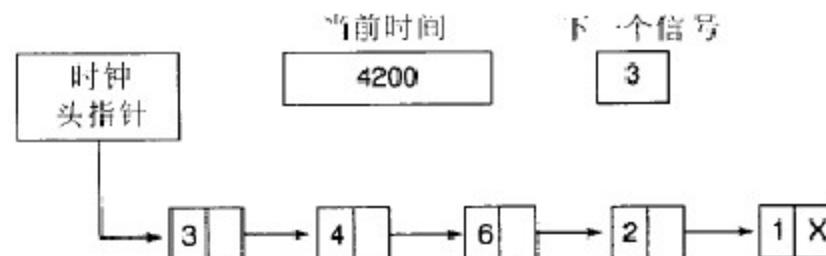
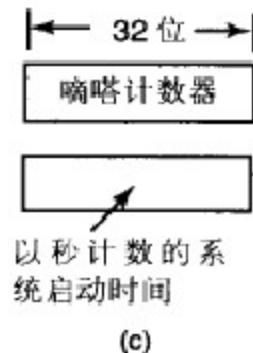


图 3-25 用一个时钟模拟多个定时器

# Minix中的时钟驱动实现

## ❖ 核心函数: **kernel\clock.c(11000行)**

- **clock\_task**: 循环等待消息的处理过程
- **syn\_clock\_task**: 自阻塞机制的处理过程
- 各种功能函数的实现

## ❖ 基于中断原理和进程通信原理的系统调用实现

- 用户进程发出系统调用指令——软件中断机制
- 时钟滴答到达时间片数目——硬件中断机制
- 定时器与同步闹钟的使用——信号传送机制 (**MM中转**)

# 内容提要

- ❖ 代码结构组织与数据结构定义
- ❖ 专题**1**——操作系统的启动过程
- ❖ 专题**2**——操作系统的进程管理
- ❖ 专题**3**——操作系统的设备管理
- ❖ 专题**4**——操作系统的内存管理
- ❖ 专题**5**——操作系统的文件管理

# Minix中的内存服务器管理

## ❖ 内存服务器的启动和初始化

- Boot引导程序将内存服务器的可执行文件放入正确的位置
- TaskTab中定义与内存服务器相关的表项
- Main函数中初始化进程表项，填充服务器进程队列
- restart所有系统任务和服务器进程，轮转到MM进程时进行初始化

## ❖ 内存服务器的主要任务

- 负责处理涉及到内存的重要系统调用
- 负责管理用户内存空间（分配与回收）
- 负责与内核（sys\_task）通信，实现“决策+机构”的分离

## ❖ 内存服务器的具体实现

- 独立的主程序——运行在非核心空间
- 处理多个系统调用（图4-34）
- 管理进程表和空洞表

## ❖ 对内存服务器实现机制的了解

- 清晰掌握内存空洞表的管理机制
- 结合进程表概念、进程通信、进程调度的基本知识，了解各种系统调用的操作细节

***Thanks for your time!***  
***Questions & Answers***

# 头文件的定义与组织

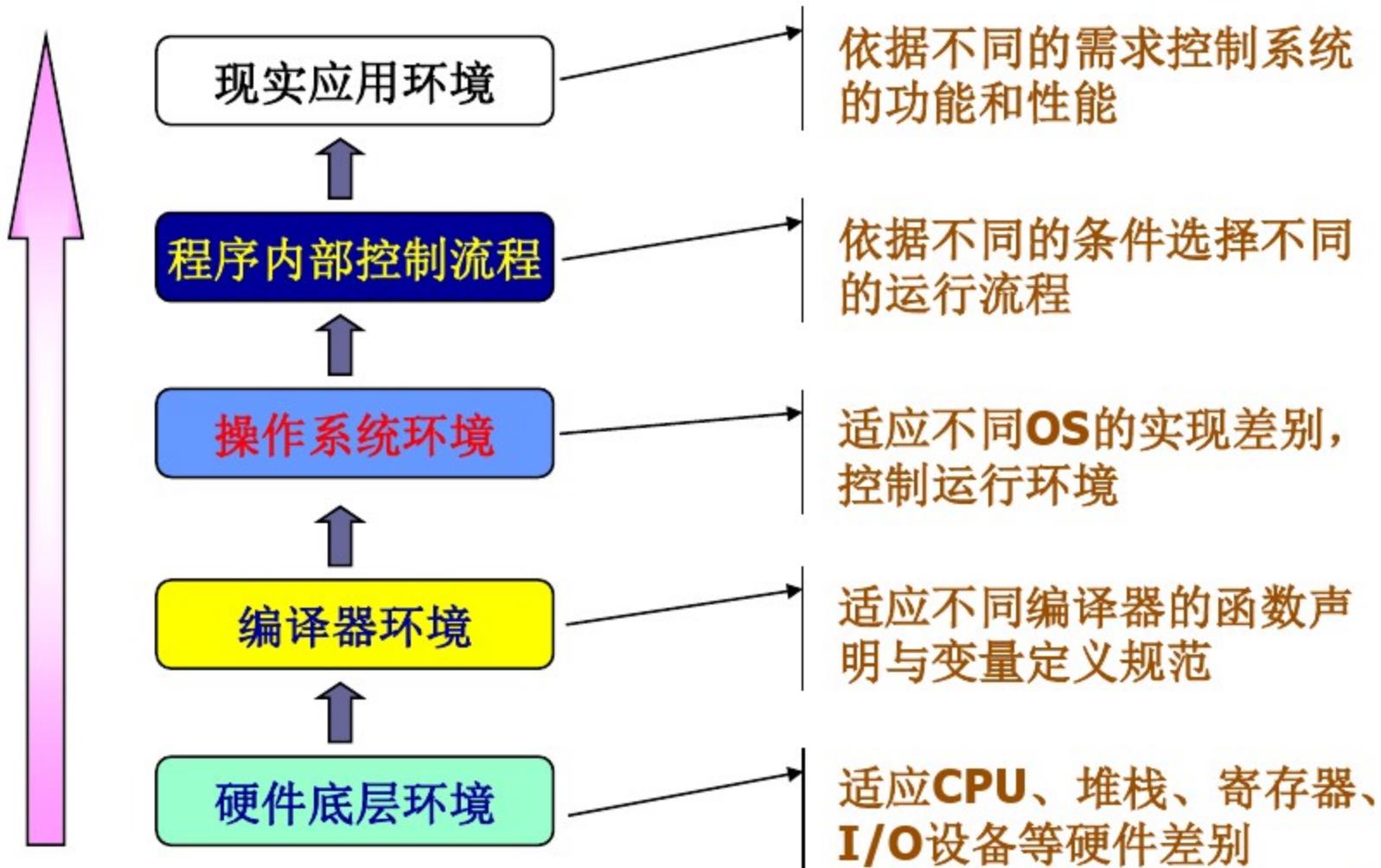
## ❖ 头文件有哪些功能？

- **功能1：** 定义数据结构、定义函数接口
- **功能2：** 宏定义，例如常量、错误代码
- **功能3：** 设定编译开关，控制运行过程
- **功能4：** 设定硬件开关，适应硬件环境
- **功能5：** 设定配置开关，限定功能范围

## ❖ 头文件的用法举例

# 软件的体系结构观点

完整的软件系统设计必须涵盖从底层至顶层的完整流程，以体系结构的思想面对软件研发



# Minix的公共头文件组织

- ❖ 适应底层硬件环境
  - 如何区别**CPU**的差别?
- ❖ 适应编译环境
  - 如何遵循**ANSI**、**POSIX**等标准，适应不同编译环境?
- ❖ 控制运行流程
  - 如何设定内部开关，控制有效范围和运行流程?
- ❖ 适应应用环境
  - 如何实现可移植性，同时提供用户配置接口?
- ❖ 实现基本功能
  - 为保持系统运行稳定和性能卓越而进行的定义与实现

# 与硬件环境相关的头文件

- ❖ 针对**CPU**和设备驱动的宏定义
  - **include/minix/config.h** (2600行)
- ❖ 针对键盘布局、硬盘分区的头文件
  - **include/minix/keymap.h**、**partition.h**
- ❖ 针对**IBM**硬件平台的专有头文件
  - **include/ibm/diskparm.h**, **partition.h**

# 与编译环境相关的头文件

- ❖ 遵循**ANSI**规范的头文件

- **include/ansi.h**、**stdlib.h**

- ❖ 遵循**POSIX**规范的头文件

- **include/unistd.h**、**string.h**、**signal.h**、**fcntl.h**、**termios.h**
  - **include/sys/types.h**、**ioctl.h**、**sigcontext.h**、**ptrace.h**、**stat.h**、**dir.h**、**wait.h**

# 与OS系统实现相关的头文件

- ❖ 基本数据类型与系统限制
  - **include/limits.h, errno.h**
- ❖ 系统实现所需头文件
  - **include/a.out.h**——可执行文件格式
- ❖ 系统相关的数据类型和内部调用
  - **include/minix/const.h、type.h、syslib.h、callnr.h、com.h、boot.h**