第二章　进程管理

翁楚良

https://chuliangweng.github.io

2023 春　ECNU

第二章进程管理 提纲

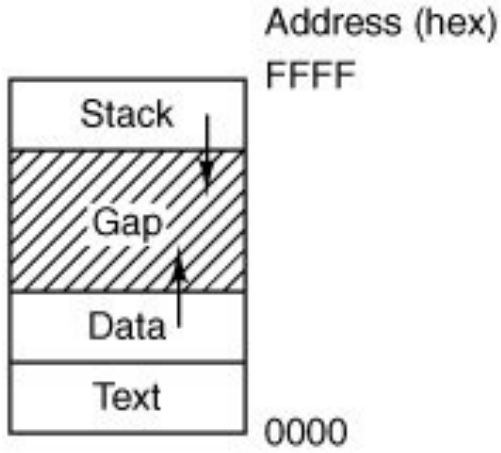
- 2.1 进程
- 2.2 进程间通信
- 2.3 经典并发问题
- 2.4 进程调度
- 2.5 MINIX3进程概述
- 2.6 MINIX3进程实现
- 2.7 MINIX3系统任务
- 2.8 MINIX3时钟任务

2.1 进程

- 进程模型
- 进程创建和终止
- 进程层次结构
- 进程状态及转换
- 进程的实现
- 线程及实现

进程模型

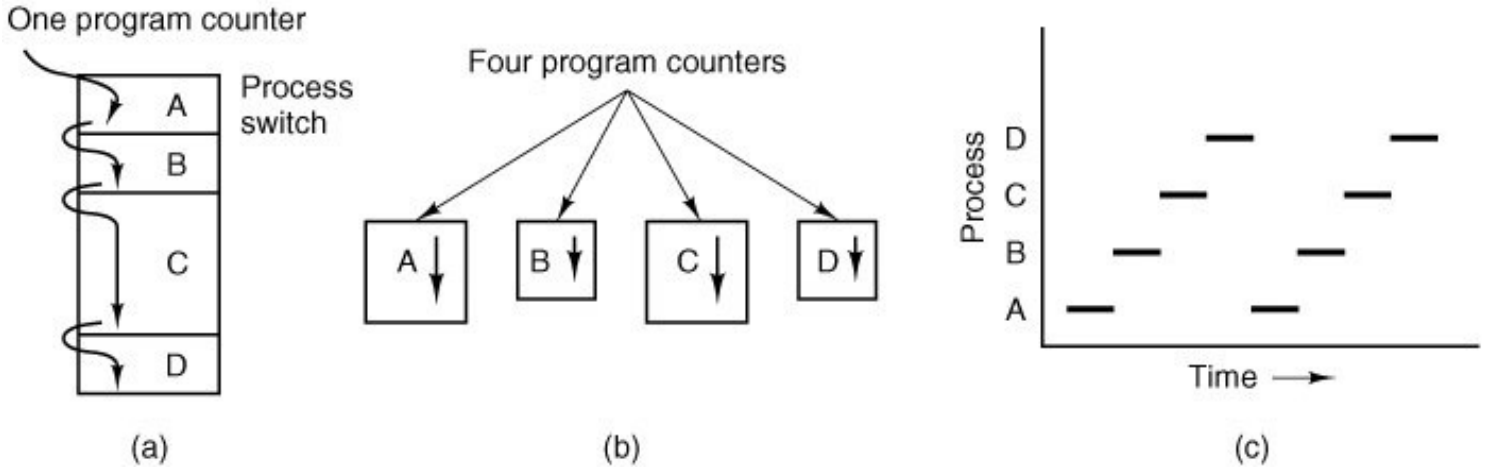
- 进程是一个正在执行程序
  - 程序计数器、寄存器和变量的当前值
  - 包括三个段：代码段、数据段、栈段



- 逻辑上：运行在属于自己的虚CPU
- 物理上：多个进程通过切换共享物理CPU
- 一个进程是某种类型的一个活动，它有程序、输入、输出、及状态。
- 单个处理机被若干进程共享，它使用某种调度算法决定何时停止一个进程的工作，并转而为另一个进程提供服务。

进程切换示例

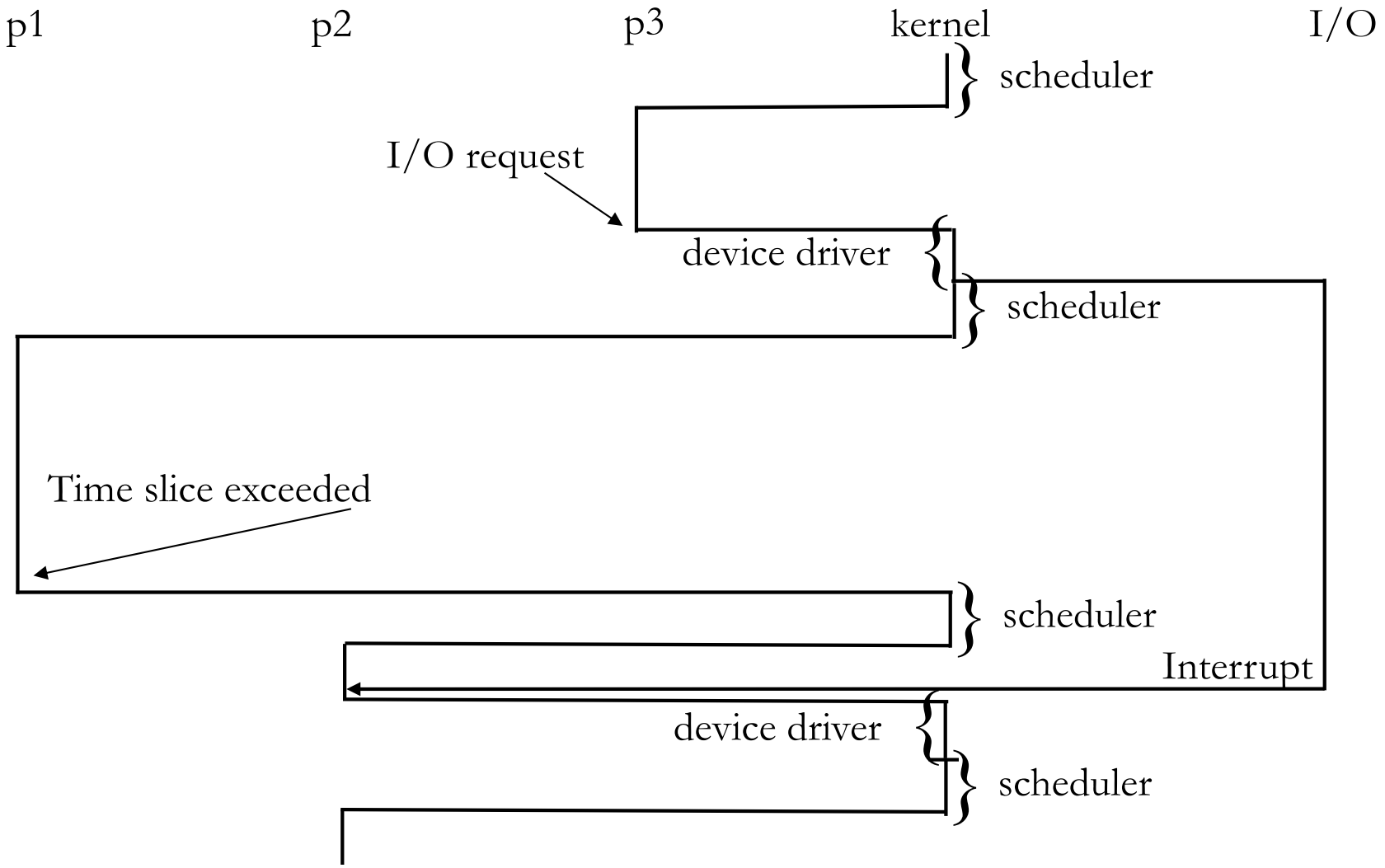
- 每个进程拥有自己的逻辑程序计数器 (控制流)
- 单CPU系统只有一个物理程序计数器
- 当进程被载入运行时，其将该进程的逻辑计数器装入到物理程序计数器中
- 当进程终止当前运行时，将物理程序计数器中值保存在该进程的逻辑计数器中。



上下文切换(context switch)

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- This is called context switch.
- The time it takes is dependent on hardware support.
- Context-switch time is overhead; the system does no useful work while switching.

进程切换举例


p1
p2
p3
kernel
I/O
scheduler
I/O request
device driver
scheduler
Time slice exceeded
scheduler
Interrupt
device driver
scheduler

模式切换(Mode Switching)

- It may happen that an interrupt does not produce a context switch.
- The control can just return to the interrupted program.
- Then only the processor state information needs to be saved on stack.
- This is called mode switching (user to kernel mode when going into Interrupt Handler).
- Less overhead: no need to update the PCB like for context switching.

●用户态时不可直接访问受保护的OS代码；
●核心态时执行OS代码，可以访问全部进程空间。

进程与程序的区别

- 进程是动态的，程序是静态的：程序是有序代码的集合；进程是程序的执行。通常进程不可在计算机之间迁移；而程序通常对应着文件、静态和可以复制。
- 进程是暂时的，程序的永久的：进程是一个状态变化的过程，程序可长久保存。
- 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块（即进程状态信息）。
- 进程与程序的对应关系：通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可包括多个程序。

2.1 进程

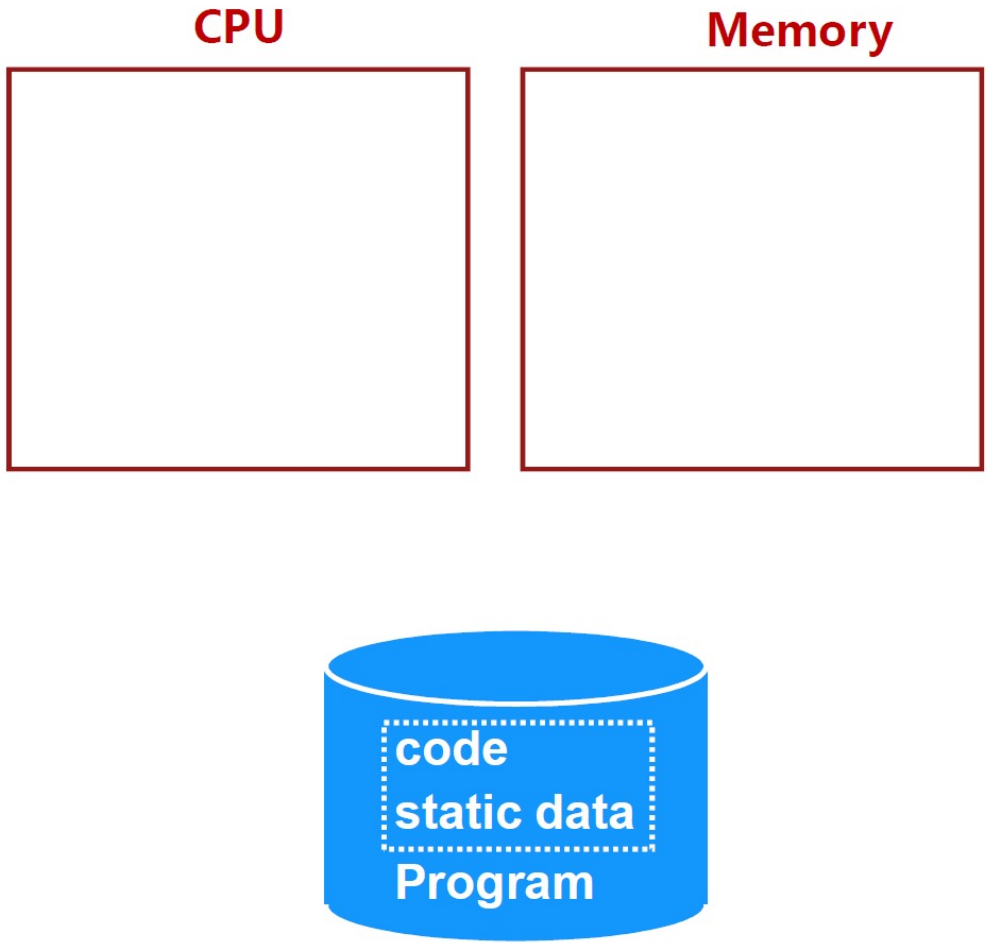
- 进程模型
- 进程创建和终止
- 进程层次结构
- 进程状态及转换
- 进程的实现
- 线程及实现

进程创建

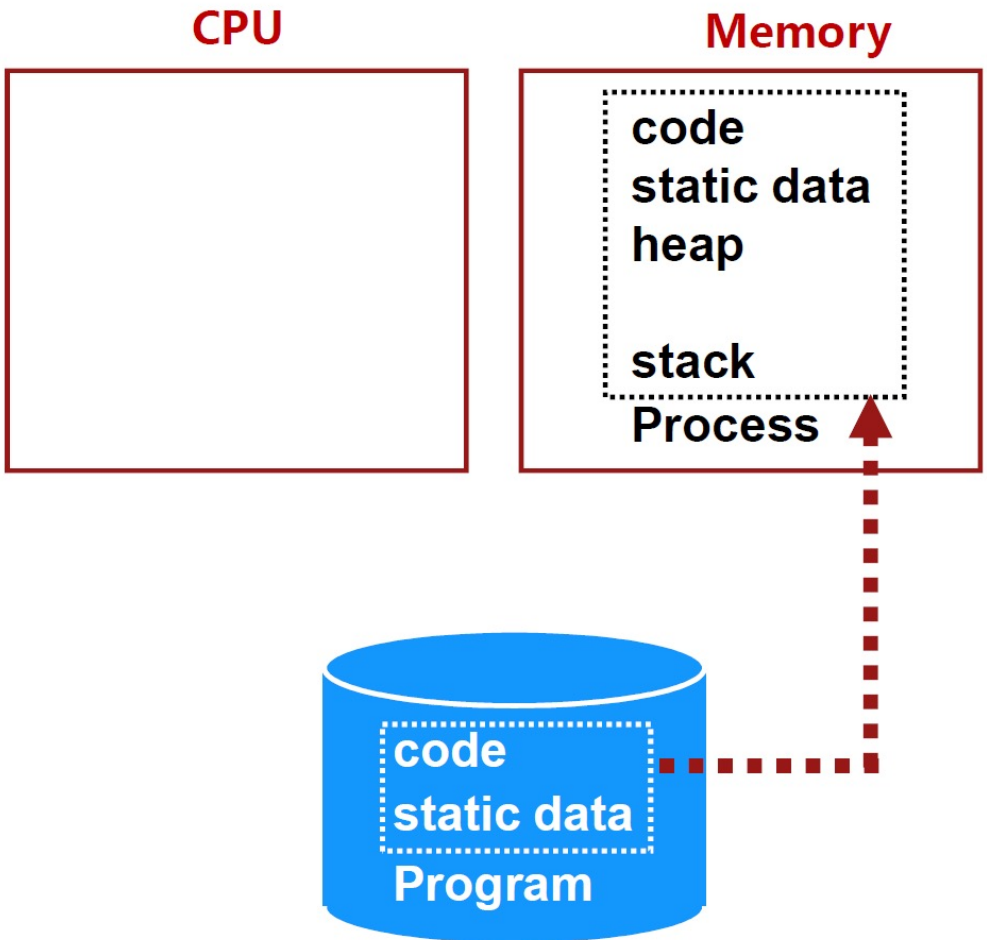
- There are four principal events that cause processes to be created:
  - System initialization.
  - Execution of a process creation system call by a running process.
  - A user request to create a new process.
  - Initiation of a batch job.

- Daemon process(守护进程)
  - Processes that stay in the background to handle some activity such as web pages, printing, and so on.

进程创建



进程创建


CPU
Memory
code
static data
heap
stack
Process
code
static data
Program

新创建进程的属性

- 拥有独立的地址空间，是原进程的一个副本
- 如果在其地址空间中修改数据，对于原进程的进程是不可见的
- 新创建的进程与原进程并发执行
  - **fork** system call creates new process.
  - **exec** system call used after **fork** to replace the process' memory image (存储映象) with a new program.

进程终止

- 进程终目原因
  - 正常退出(自愿)
  - 出错退出(自愿)
  - 严重错误(非自愿)
  - 被其它进程终止(非自愿)

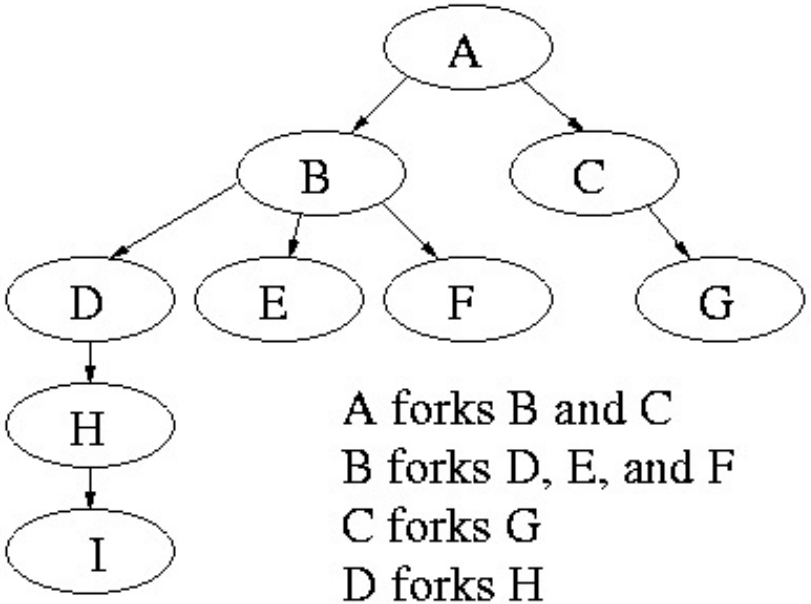
- 在UNIX系统中，系统调用kill和exit用于终止进程
  - kill 向进程发送信号，可以终止进程
  - exit用于进程终止自身

2.1 进程

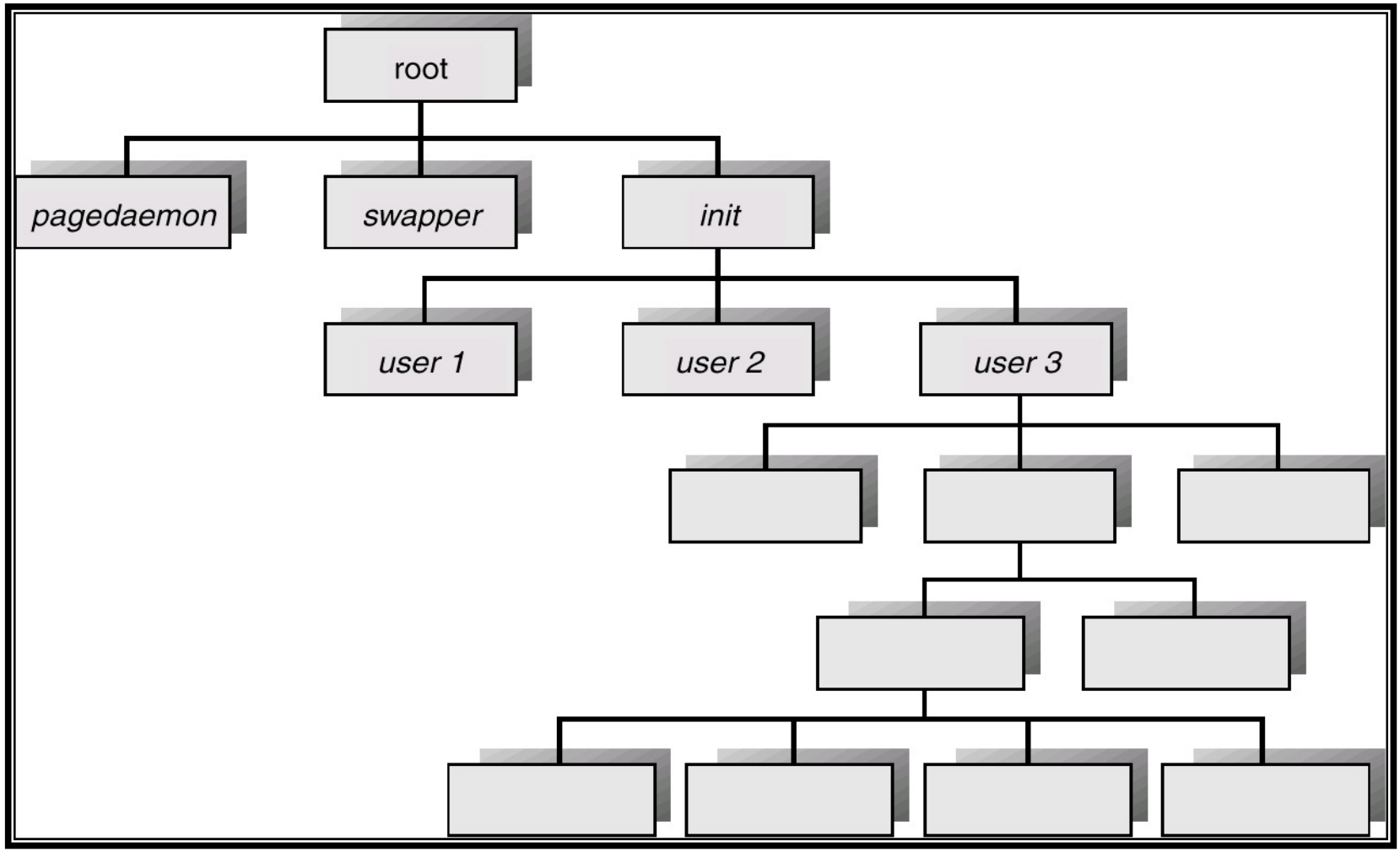
- 进程模型
- 进程创建和终止
- 进程层次结构
- 进程状态及转换
- 进程的实现
- 线程及实现

进程层次结构

- 进程被创建之后，进程间存在一定的关联性质
  - parent process – 父进程
  - child process – 子进程
- 被创建的进程可以创建自己的子进程
  - 每个进程只有一个父进程
  - 一个进程可以有多个子进程
  - 进程间的关系可以用树结构表示，即进程树



Processes Tree on a UNIX System



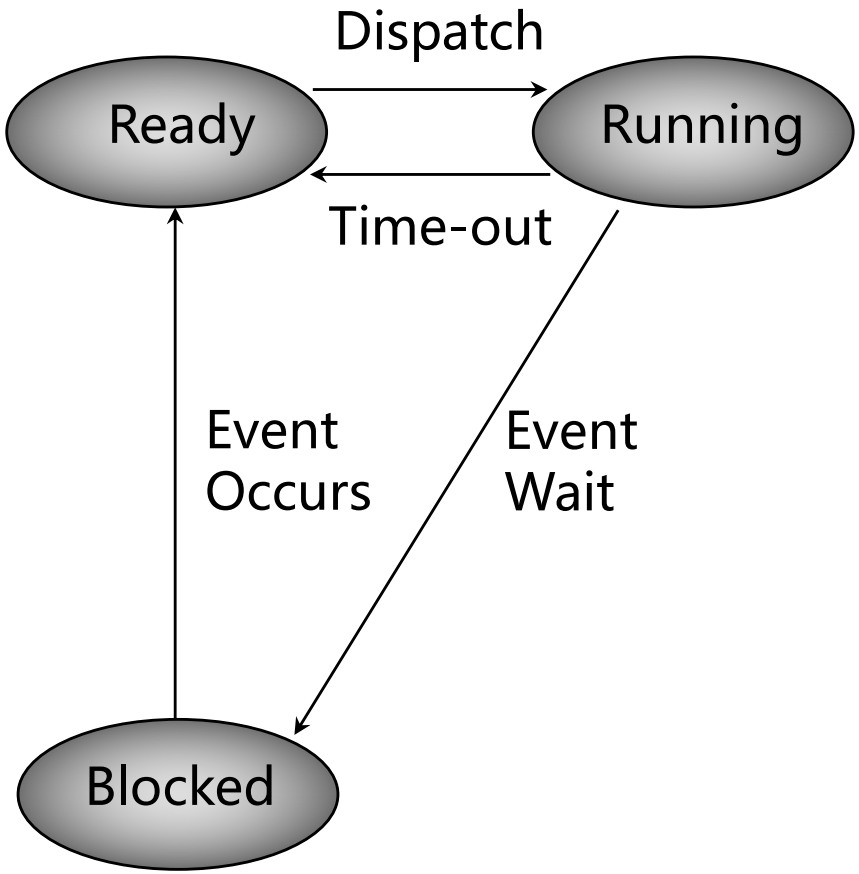
2.1 进程

- 进程模型
- 进程创建和终止
- 进程层次结构
- 进程状态及转换
- 进程的实现
- 线程及实现

Process States

- Let us start with three states:
  - Running state -
    - The process that gets executed (single CPU).
  - Ready state -
    - any process that is ready to be executed.
  - Blocked/Waiting state -
    - when a process cannot execute until some event occurs (e.g., the completion of an I/O).

A Three-state Process Model



Process Transitions (1)

- Ready --> Running
  - When it is time, the dispatcher selects a new process to run.
- Running --> Ready
  - the running process has expired his time slot.
  - the running process gets interrupted because a higher priority process is in the ready state.

Process Transitions (2)

- Running --> Blocked
  - When a process requests something for which it must wait:
    - a service that the OS is not ready to perform.
    - an access to a resource not yet available.
    - initiates I/O and must wait for the result .
    - waiting for a process to provide input.
- Blocked --> Ready
  - When the event for which it was waiting occurs.

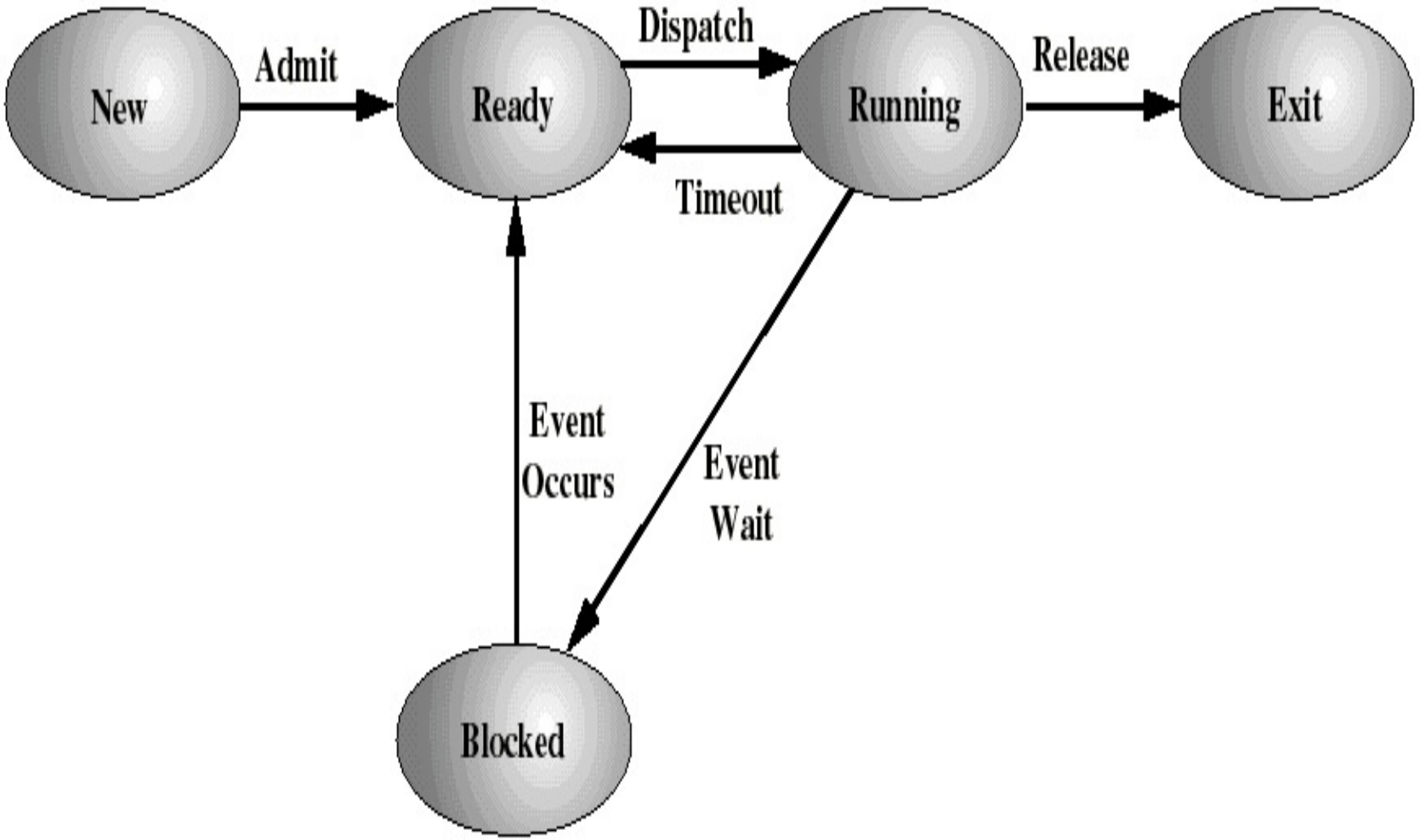
Other Useful States (1)

- New state -
  - OS has performed the necessary actions to create the process:
    - has created a process identifier.
    - has created tables needed to manage the process.
  - but has not yet committed to execute the process (not yet admitted):
    - because resources are limited.

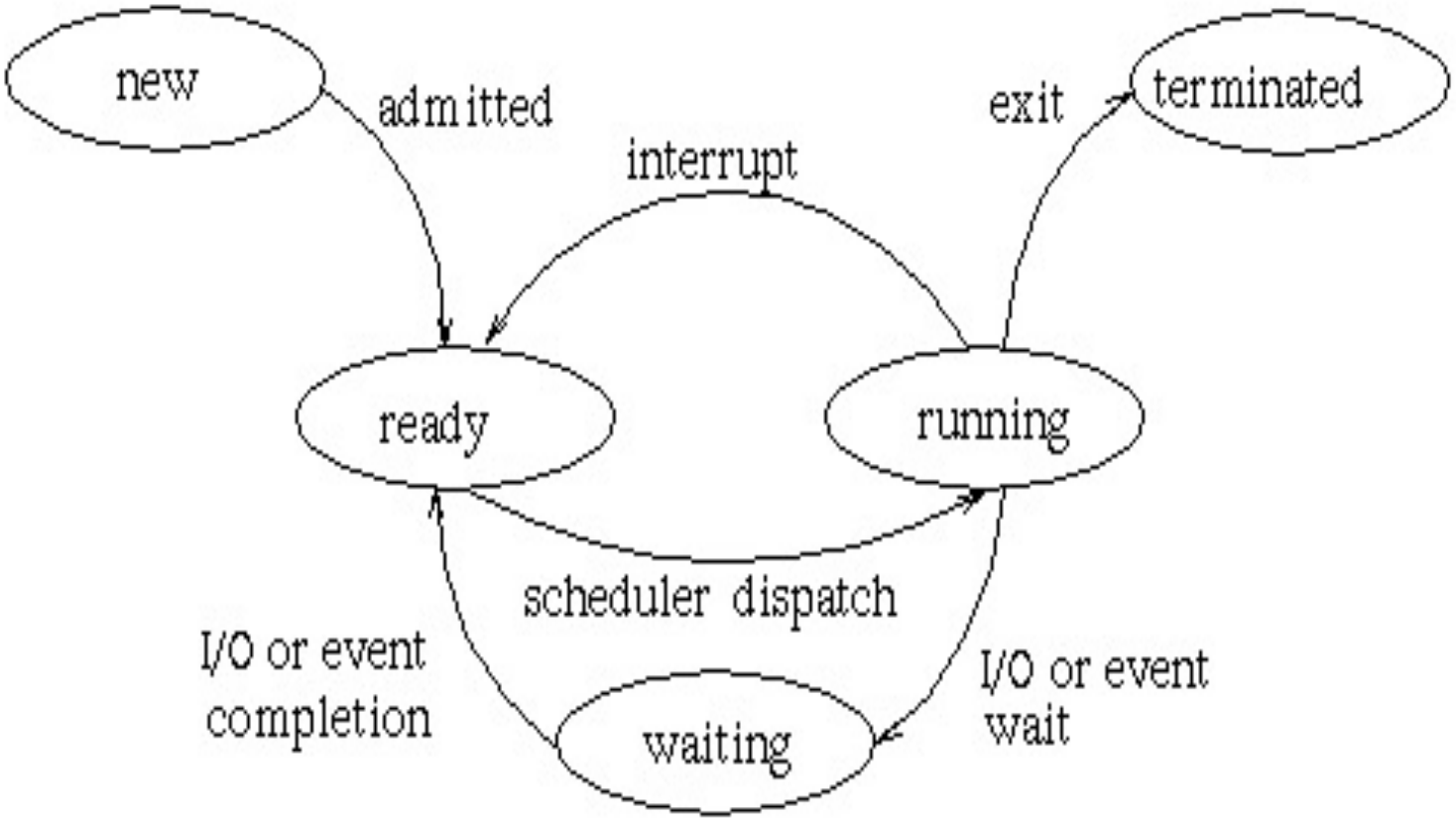
Other Useful States (2)

- Exit/Terminated state -
  - Program termination moves the process to this state.
  - It is no longer eligible for execution.
  - Tables and other info are temporarily preserved for auxiliary program -
    - Ex: accounting program that cumulates resource usage for billing the users.
- The process (and its tables) gets deleted when the data is no more needed.

A Five-state Process Model

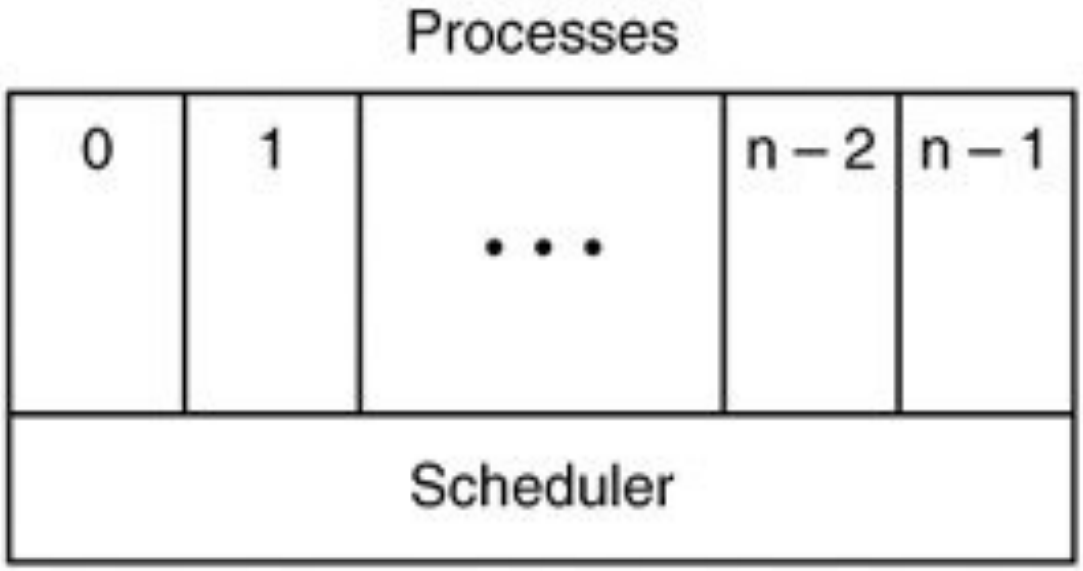


Another view of Five-state Process Model



操作系统视图

- 由进程构成的操作系统
- 最低层处理中断与调度
- 上层是串行程序



2.1 进程

- 进程模型
- 进程创建和终止
- 进程层次结构
- 进程状态及转换
- 进程的实现
- 线程及实现

进程的实现

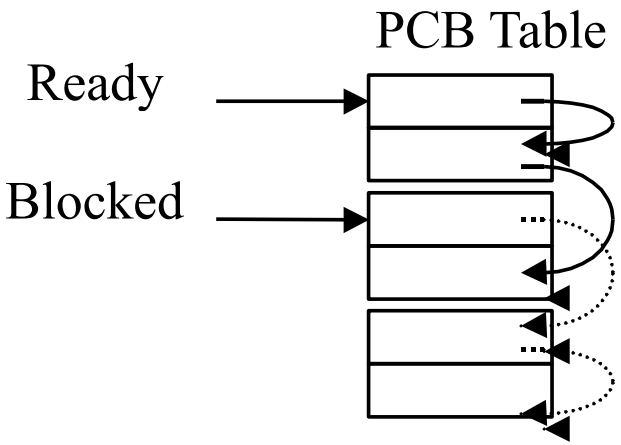
- 进程控制块(PCB, process control block)
  - 进程控制块是由OS维护的用来记录进程相关信息的一块内存
  - 每个进程在OS中的登记表项（可能有总数目限制），OS据此对进程进行控制和管理（PCB中的内容会动态改变）
  - 处于核心段，通常不能由应用程序自身的代码来直接访问，而要通过系统调用，或通过UNIX中的进程文件系统(/proc)直接访问进程映象(image)。文件名为进程标识（如：00316），权限为创建者可读写

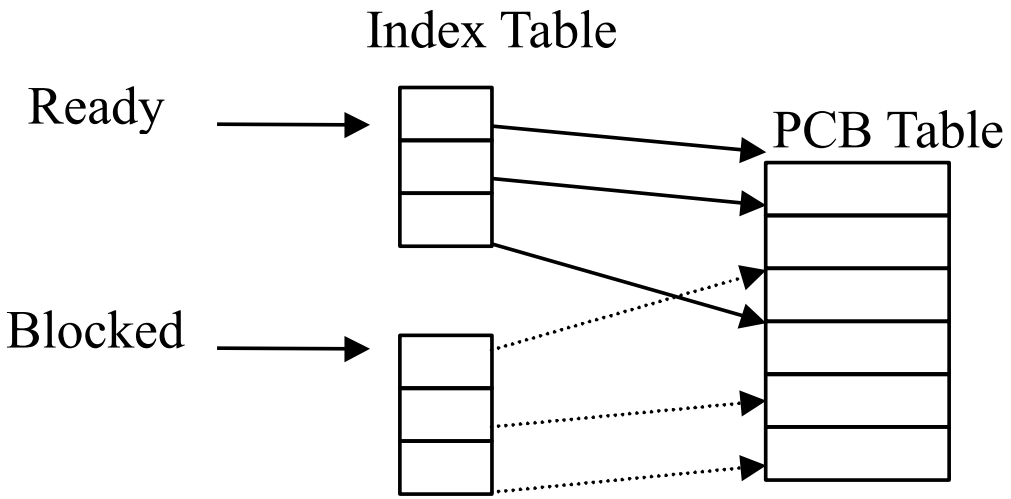
进程控制块的内容

- 进程描述信息：
  - 进程标识符(process ID)，唯一，通常是一个整数；
  - 进程名，通常基于可执行文件名（不唯一）；
  - 用户标识符(user ID)；进程组关系(process group)
- 进程控制信息：
  - 当前状态；
  - 优先级(priority)；
  - 代码执行入口地址；
  - 程序的外存地址；
  - 运行统计信息（执行时间、页面调度）；
  - 进程间同步和通信；阻塞原因
- 资源占用信息：虚拟地址空间的现状、打开文件列表
- CPU现场保护结构：寄存器值（通用、程序计数器PC、状态PSW，地址包括栈指针）

PCB的组织方式

- 链表：同一状态的进程其PCB成一链表，多个状态对应多个不同的链表
  - 各状态的进程形成不同的链表：就绪链表、阻塞链表
- 索引表：同一状态的进程归入一个index表（由index指向PCB），多个状态对应多个不同的index表
  - 各状态的进程形成不同的索引表：就绪索引表、阻塞索引表





MINIX3进程控制块

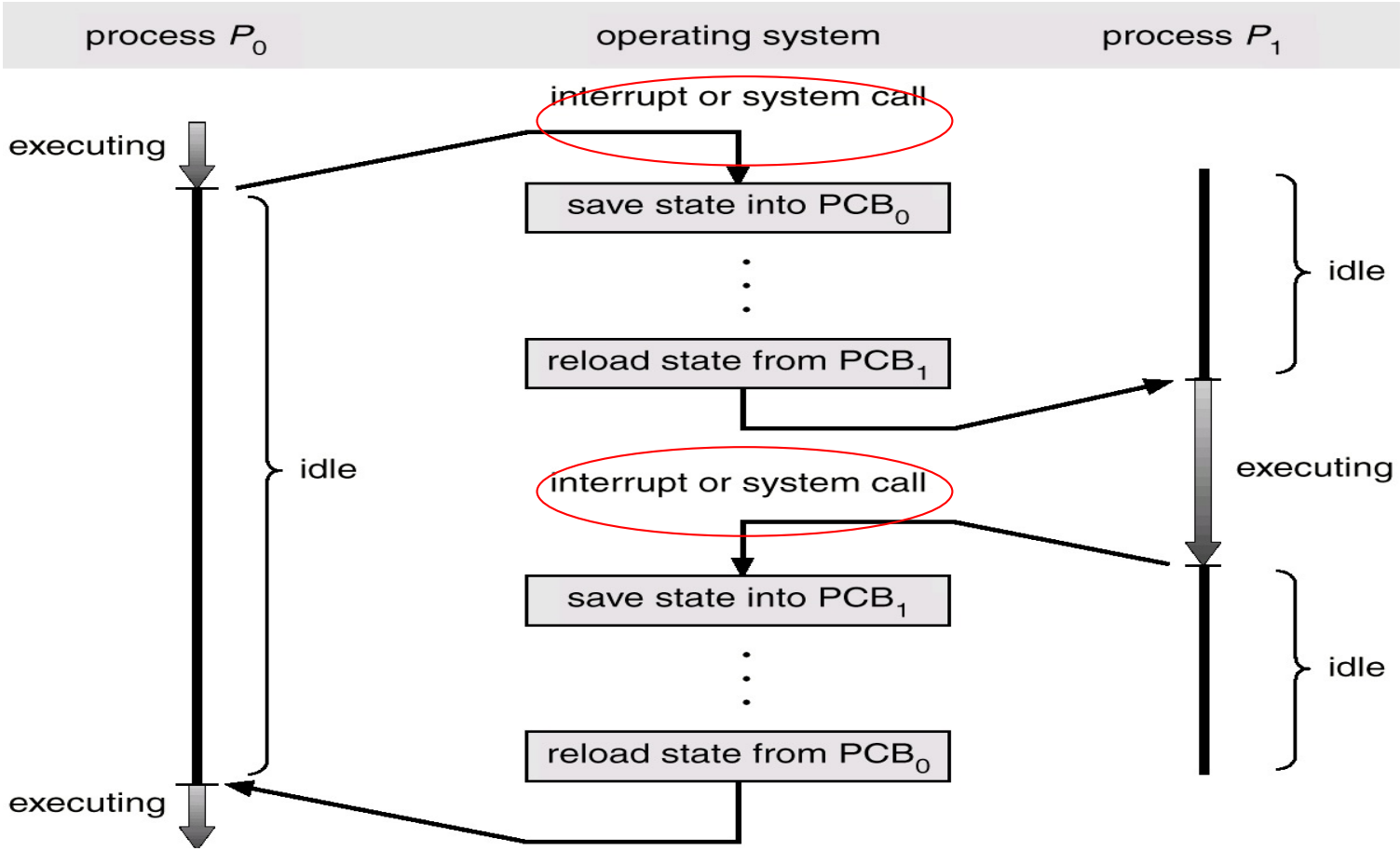
- 在MINIX3中进程通信、内存管理和文件管理是由系统中的几个模块分别处理，左图列出进程控制块(进程表)一些重要的域。

Kernel	Process management	File management
Registers	Pointer to text segment	UMASK mask
Program counter	Pointer to data segment	Root directory
Program status word	Pointer to bss segment	Working directory
Stack pointer	Exit status	File descriptors
Process state	Signal status	Real id
Current scheduling priority	Process ID	Effective UID
Maximum scheduling priority	Parent process	Real GID
Scheduling ticks left	Process group	Effective GID
Quantum size	Children's CPU time	Controlling tty
CPU time used	Real UID	Save area for read/write
Message queue pointers	Effective UID	System call parameters
Pending signal bits	Real GID	Various flag bits
Various flag bits	Effective GID	
Process name	File info for sharing text	
	Bitmaps for signals	
	Various flag bits	
	Process name	

Linux 0.11 进程控制块

```
struct task_struct {
        long state;                     //任务的运行状态（-1 不可运行，0 可运行(就绪)，>0 已停止）。
        long counter;                   // 任务运行时间计数(递减)（滴答数），运行时间片。
        long priority;                  // 运行优先数。任务开始运行时 counter=priority,越大运行越长。
        long signal;                    // 信号。是位图，每个比特位代表一种信号，信号值=位偏移值+1。
        struct sigaction sigaction[32]; // 信号执行属性结构，对应信号将要执行的操作和标志信息。
        long blocked;                   // 进程信号屏蔽码（对应信号位图）。
        int exit_code;                  // 任务执行停止的退出码，其父进程会取。
        unsigned long start_code;       // 代码段地址。
        unsigned long end_code;         // 代码长度（字节数）。
        unsigned long end_data;         // 代码长度 + 数据长度（字节数）。
        unsigned long brk;              // 总长度（字节数）。
        unsigned long start_stack;      // 堆栈段地址。
        long pid;                       // 进程标识号(进程号)。
        long father;                  // 父进程号。
        long pgrp;                    // 进程组号。
        long session;                 // 会话号。
        long leader;                  // 会话首领。
        unsigned short uid;           // 用户标识号（用户 id）。
        unsigned short euid;          // 有效用户 id。
        unsigned short suid;          // 保存的用户 id。
        unsigned short gid;           // 组标识号（组 id）。
        unsigned short egid;          // 有效组 id。
        unsigned short sgid;          // 保存的组 id。
        long alarm;                   // 报警定时值（滴答数）。
        long utime;                   // 用户态运行时间（滴答数）。
        long stime;                   // 系统态运行时间（滴答数）。
        long cutime;                  // 子进程用户态运行时间。
        long cstime;                  // 子进程系统态运行时间。
        long start_time;              // 进程开始运行时刻。
        unsigned short used_math;     // 标志：是否使用了协处理器。
        int tty;                      // 进程使用 tty 终端的子设备号。-1 表示没有使用。
        unsigned short umask;         // 文件创建属性屏蔽位。
        struct m_inode * pwd;         // 当前工作目录 i 节点结构指针。
        struct m_inode * root;        // 根目录 i 节点结构指针。
        struct m_inode * executable; // 执行文件 i 节点结构指针。
        unsigned long close_on_exec; // 执行时关闭文件句柄位图标志。（参见 include/fcntl.h）
        struct file * filp[NR_OPEN]; // 文件结构指针表，最多 32 项。表项号即是文件描述符的值。
        struct desc_struct ldt[3];    // 局部描述符表。0-空，1-代码段 cs，2-数据和堆栈段 ds&ss。
        struct tss_struct tss;        // 进程的任务状态段信息结构。
};
```

进程切换过程实现


process P_0
operating system
process P_1
executing
interrupt or system call
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing