OPERATING SYSTEMS

Introduction to Process

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

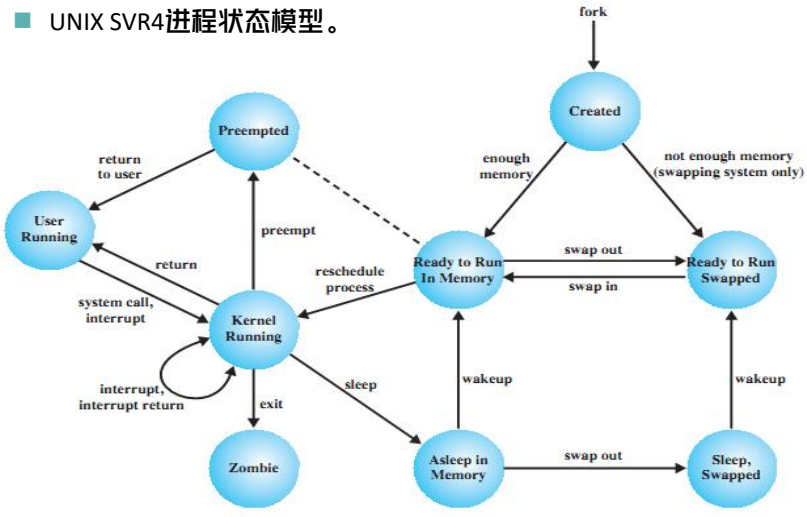
- **目录**
  - 基本概念
  - 过程表和过程控制块
  - 过程状态和转换
  - 进程操作
    - 进程创建
    - 进程终止
  - Unix和Linux示例
  - **进程调度**
  - **进程切换**

Unix and Linux Examples

- **UNIX SVR4（System V Release 4，最新版UNIX）进程状态**
  - **用户运行** - 在用户模式下执行。
  - **内核运行** - 在内核模式下执行。
  - **在内存中准备运行** - 准备在内核调度它时立即运行。
  - **在内存中休眠** - 在事件发生之前无法执行；进程在主存中（阻塞状态）。
  - **准备交换运行** - 进程已经准备好运行，但交换程序必须将进程交换到主内存中，然后内核才能安排它执行
  - **已交换休眠** - 进程正等待事件，已交换到辅助存储器（阻塞状态）
  - **抢占Preempted** - 进程正在从内核模式返回到用户模式，但内核抢占它并执行进程切换以调度另一个进程。
  - **已创建** - 进程是新创建的，尚未准备好运行。
  - **僵尸** - 进程不再存在，但它会留下一条记录供其父进程收集。

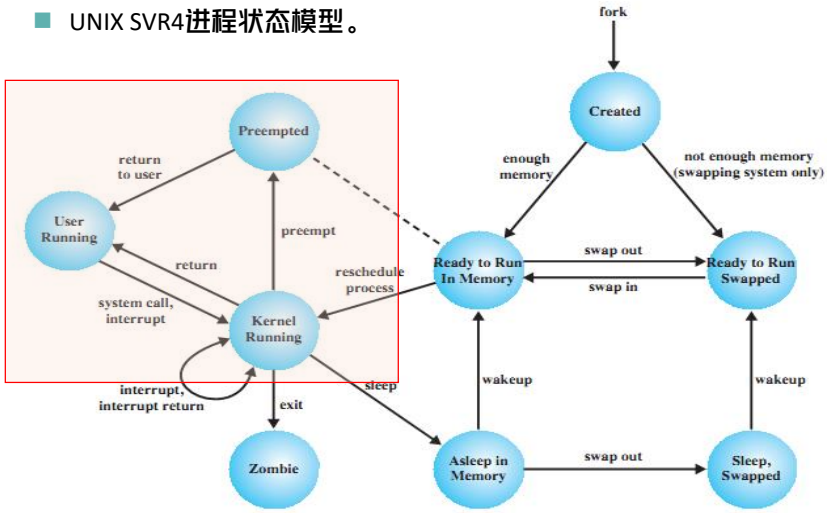
Unix and Linux Examples

- **UNIX SVR4进程状态**
  - UNIX SVR4进程状态模型。



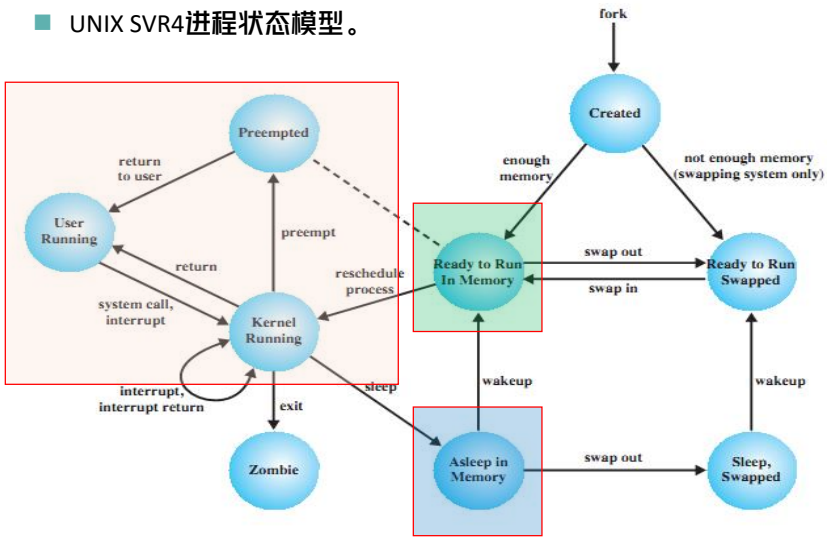
Unix and Linux Examples

- **UNIX SVR4进程状态**
  - UNIX SVR4进程状态模型。



Unix and Linux Examples

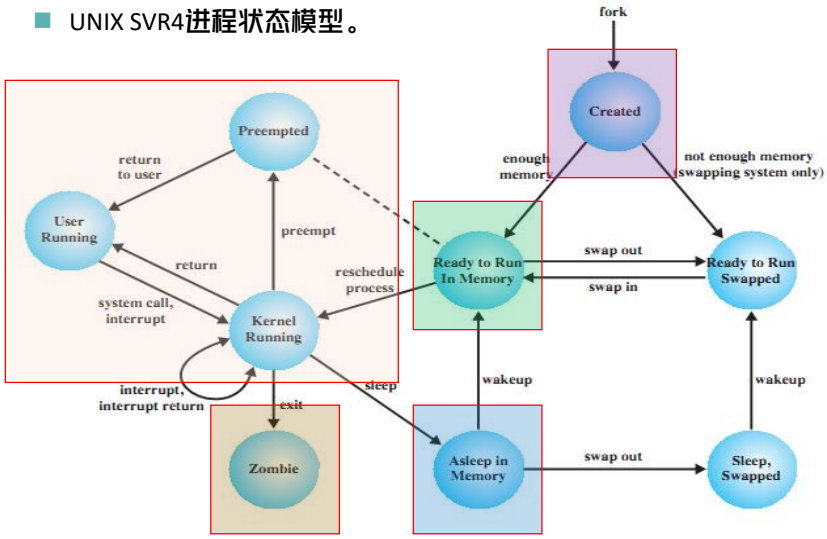
- **UNIX SVR4进程状态**
  - UNIX SVR4进程状态模型。



三态转换

Unix and Linux Examples

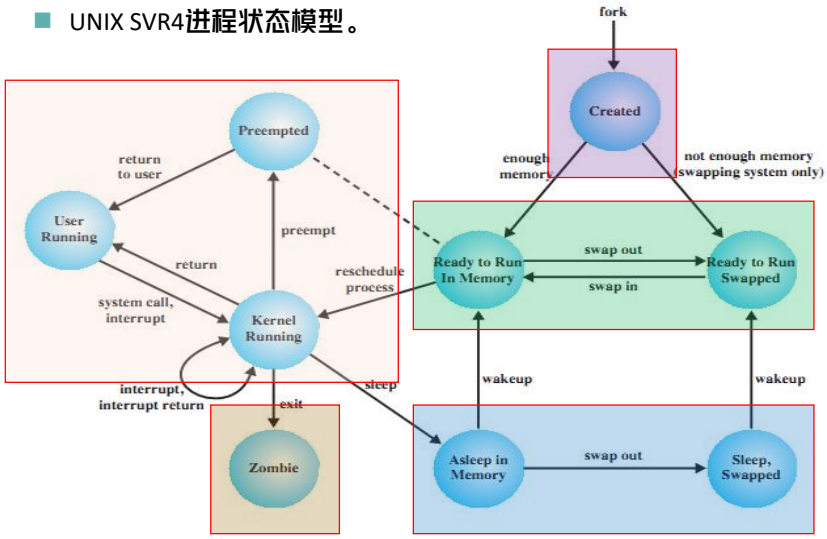
- **UNIX SVR4进程状态**
  - UNIX SVR4进程状态模型。



五态转换

Unix and Linux Examples

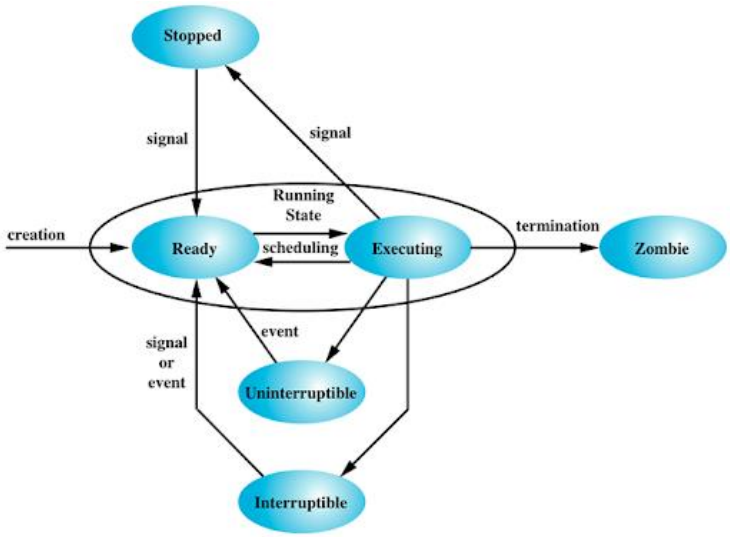
- **UNIX SVR4进程状态**
  - UNIX SVR4进程状态模型。



带交换的五态转换

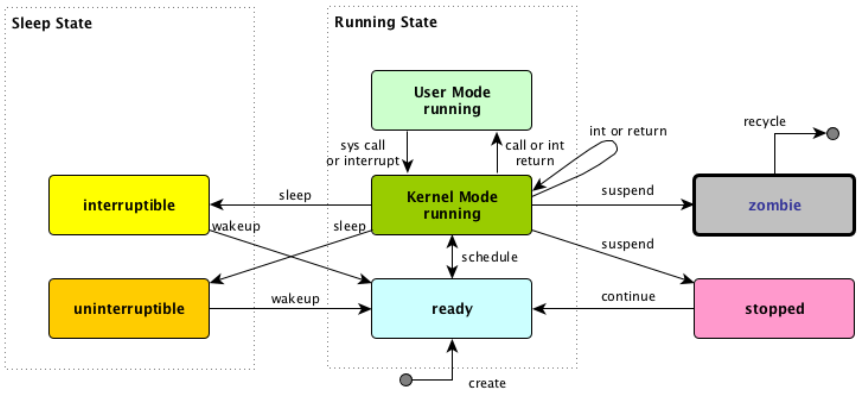
Unix and Linux Examples

- **Linux中的进程表示**
  - Linux进程状态。



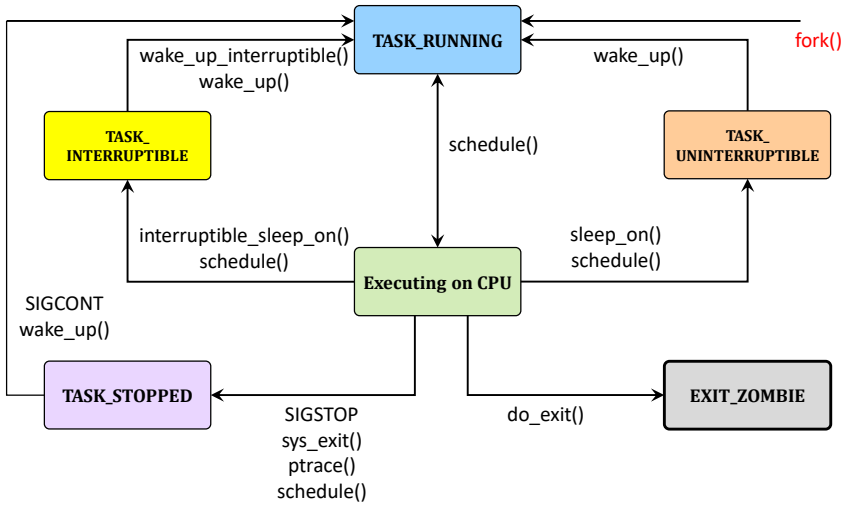
Unix and Linux Examples

- **Linux中的进程表示**
  - Linux进程状态。



Unix and Linux Examples

- **Linux中的进程表示**
  - Linux进程状态。



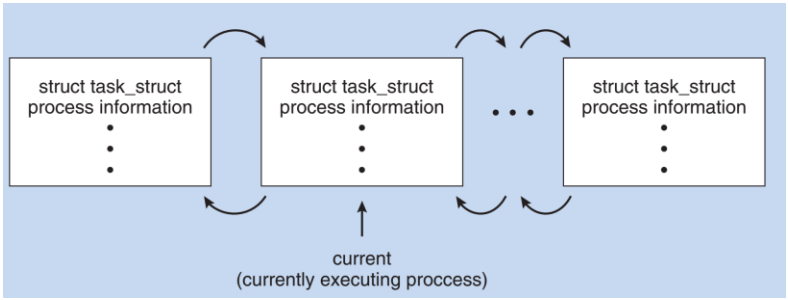
Unix and Linux Examples

- **Linux中的进程表示**
  - Linux中的PCB由<Linux/sched.h>中包含的C结构体task_struct表示
    - task_struct在linux/sched.h中声明

      sudo apt install linux-source

      tar …

      vim /usr/src/linux-source-X.XX.X/include/linux/sched.h
  - Linux内核使用struct task_struct的循环双链表来存储这些进程描述符



Unix and Linux Examples

- **Linux中的进程表示**
  - task_struct包含表示进程的所有必要信息，包括
    - PID
    - 进程的状态
    - 处理器寄存器
    - 调度和内存管理信息
    - 打开的文件列表
    - 指向父进程、子进程和同级进程的指针
  - 其中一些字段包括：

```
long state; /* state of the process */
struct sched_entity se; /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Unix and Linux Examples

- **Linux中的进程表示**
  - 以下是内核2.6.15-1.2054_FC5中的几个字段，从第701行开始：（在你的Ubuntu版本中这些字段怎么样？）

```
701.struct task_struct {
702.    volatile long state; /*-1 unrunnable,0 runnable,>0 stopped */
703.    struct thread_info *thread_info;
          . . .
767.      /* PID/PID hash table linkage. */
768.    struct pid pids[PIDTYPE_MAX];
          . . .
798.    char comm[TASK_COMM_LEN]; /* executable name excluding path
```

  - volatile long类型定义的状态

```
#define TASK_RUNNING 0
#define TASK_INTERRUPTIBLE 1
#define TASK_UNINTERRUPTIBLE 2
#define TASK_STOPPED 4
#define TASK_TRACED 8 /* in tsk->exit_state */
#define EXIT_ZOMBIE 16
#define EXIT_DEAD 32 /* in tsk->state_again */
#define TASK_NONINTERACTIVE 64
```

Process Scheduling

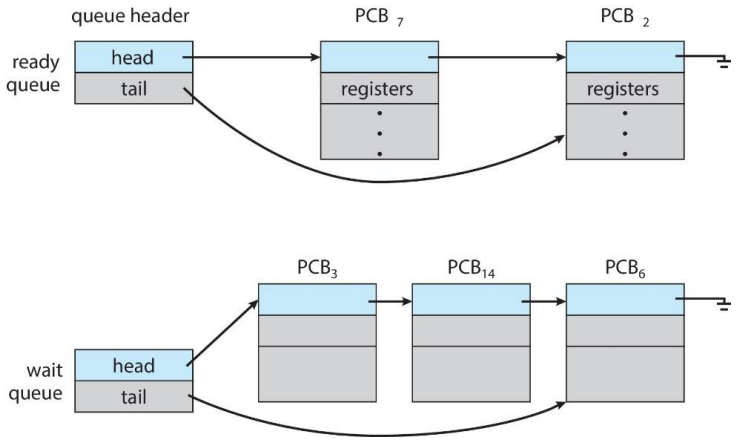
- **概述**
  - 多道程序设计的目标是让某些进程始终运行，以最大限度地提高CPU利用率。
    - 分时的目标是在进程之间频繁切换CPU，以便用户可以在每个程序运行时与之交互。
  - *进程调试器*从可用进程中选择下一个在CPU上执行的进程
    - 对于单处理器系统，只有一个正在运行的进程，其余进程必须等待CPU空闲才能重新调度

Process Scheduling

- **概述**
  - 要维护进程的*调度队列*:
    - 作业队列
      - 系统中所有进程的集合
    - 就绪队列
      - 驻留在主内存中、准备好并等待执行的所有进程的集合
    - 设备/等待队列
      - 等待I/O设备的一组进程
  - 进程在各种队列之间迁移。

Process Scheduling

- **概述**
  - 就绪和等待队列。



Process Scheduling

- **概述**
  - 大多数进程可以描述为I/O密集或CPU密集：
    - I/O密集进程
      - 做I/O的时间比计算的时间多
        - 短CPU突发
    - CPU密集进程
      - 花更多的时间做计算
        - 长CPU突发

进程调度器的类型

- 有三种类型/级别的进程调度器
  - 长期调度程序
    - 高级调度程序，或
    - 作业调度程序。
  - 中期调度程序
    - 中级调度程序，
    - 交换调度程序，或
    - 紧急调度员。
  - 短期调度程序
    - 低级调度程序，
    - CPU调度程序，
    - 微调度程序，或
    - 狭义上的进程/线程调度器

长期调度程序

- 长期进程调度器选择应将哪些程序/进程放入就绪队列。
  - 确定允许哪些程序进入系统进行处理
  - 控制多道程序设计的程度
  - 努力实现I/O密集进程和CPU密集进程的良好混合
  - 长期调度程序很少被调用
    - 秒，分钟
    - 可能很慢
- 如果允许更多进程进行处理：
  - 所有进程被阻止的可能性较小
    - 带来更好的CPU利用率
  - 每个进程占用的CPU时间较少

短期调度器

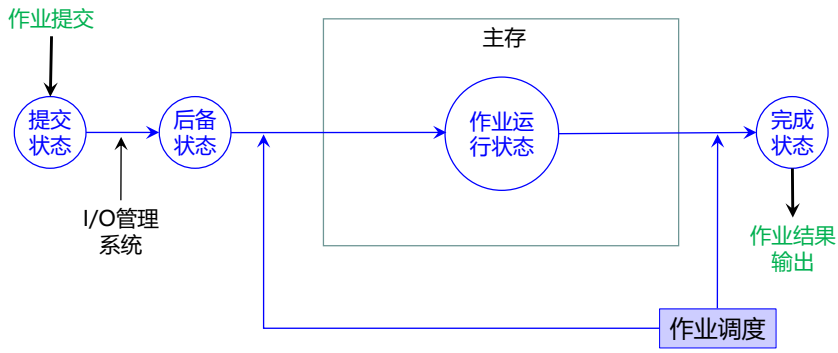
- 短期进程调度器选择下一个要执行的进程并分配CPU—也称为CPU调度(处理机调度).
  - CPU调度根据调度算法确定下一个要执行的进程。
  - 短期调度器也称为分发器*dispatcher*（它是其中的一部分），它将处理器从一个进程移动到另一个进程，并防止单个进程独占处理器时间。
  - 在可能导致选择另一个进程执行的事件上调用短期调度程序：
    - 时钟中断
    - I/O中断
    - 操作系统调用和陷阱
    - 信号
  - 短期调度程序被频繁调用
    - 毫秒
    - 务必很快

中期调度程序

- 中期进程调度器选择在系统过载时应调出的作业/进程。
  - 到目前为止，所有进程都必须（至少部分）在主内存中。
  - 即使使用虚拟内存，在主内存中保留太多进程也会降低系统性能。
  - 操作系统可能需要将一些进程交换到磁盘，然后再将它们交换回主内存。
  - 交换决策基于多道程序管理的需要。

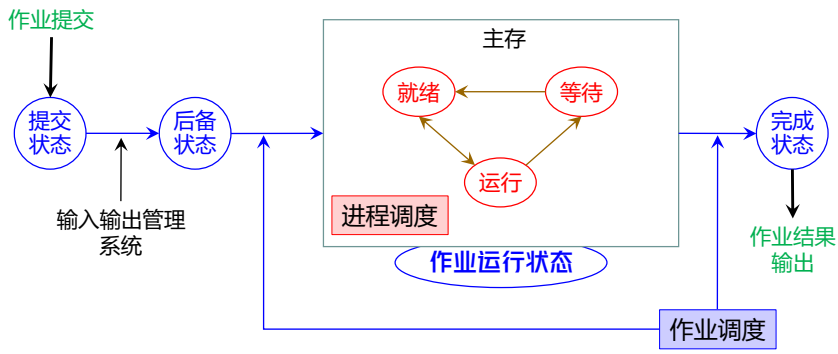
调度器的示意图

- 作业调度、交换调度和进程/线程调度。



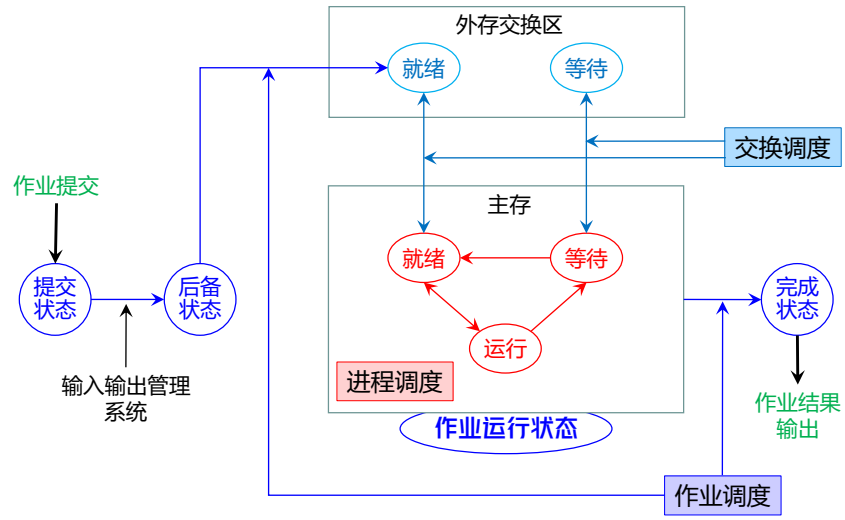
调度器的示意图

- 作业调度、交换调度和进程/线程调度。



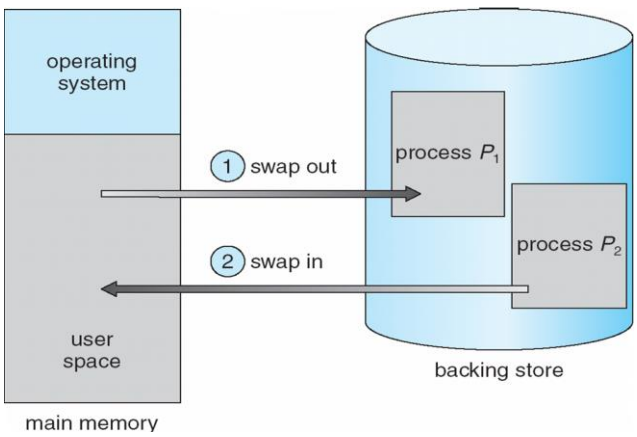
调度器的示意图

- 作业调度、交换调度和进程/线程调度。



进程交换

- 交换的示意图。

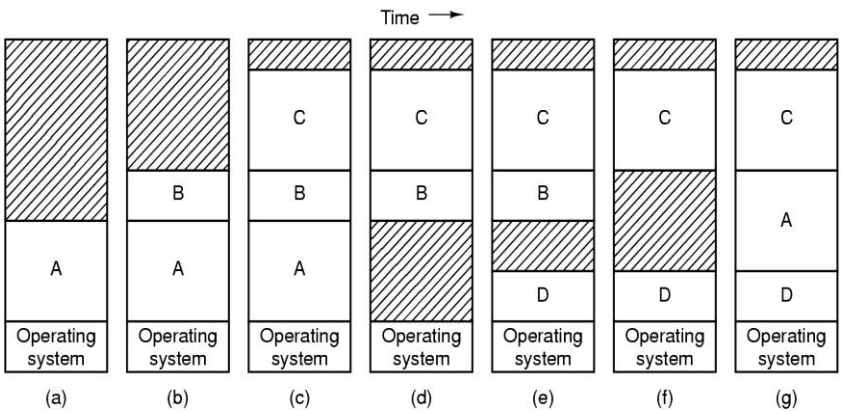


进程交换

- 交换的动态
  - 一个进程可以暂时从内存中交换到备份存储器，然后带回内存继续执行。
  - 后备存储器
    - 足够大的快速磁盘，可容纳所有用户的所有内存映像副本
    - 必须提供对这些内存映像的直接访问
  - 转入转出
    - 使用多种基于优先级的调度算法；较低优先级的进程被调出，以便可以加载和执行较高优先级的进程
  - 交换时间的主要部分是传输时间；总传输时间与交换的内存量成正比。
  - 许多系统上都有经过修改的不同版本的交换方法
    - 例如，UNIX、Linux和Windows。
  - 系统维护的就绪队列的进程，其内存映像都保存在磁盘上。

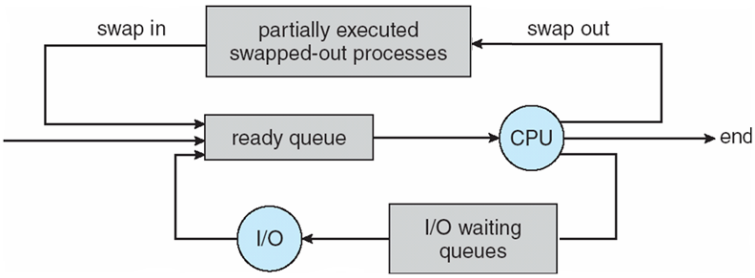
进程交换

- 交换示例。



Process Scheduling

- **进程交换**
  - 增加中期调度。

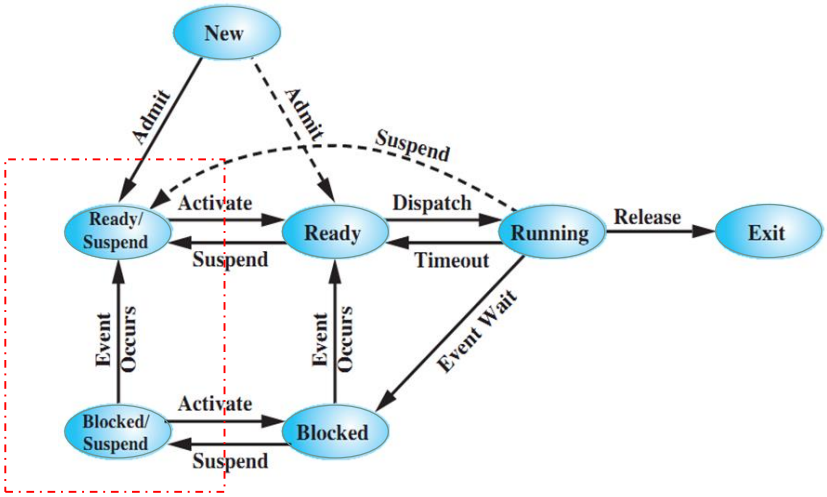


Process Scheduling

- **进程交换**
  - 交换的支持
    - 操作系统可能需要挂起一些进程，即将它们交换到磁盘，然后再交换回磁盘。
    - 可能需要增加两个新的状态：
      - 被阻止的挂起：被阻止的进程已被交换到磁盘
      - 就绪挂起：已交换到磁盘的就绪进程

Process Scheduling

- **进程调度活动**
  - 七状态进程模型

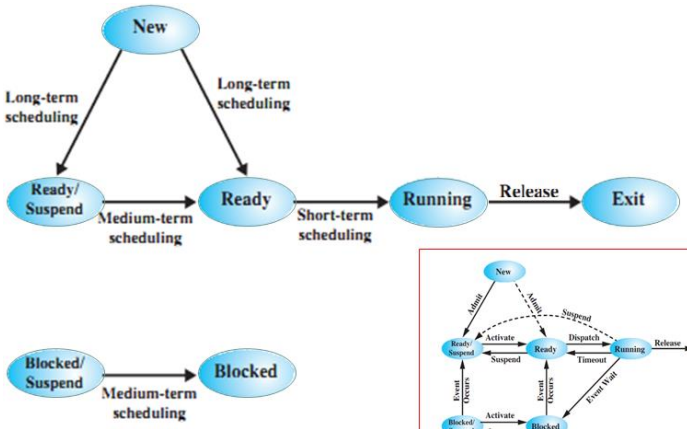


Process Scheduling

- **进程调度活动**
  - 七状态进程模型
    - 增加的状态转换
      - 阻塞 → 阻塞挂起
        - 当所有进程都被阻塞时，操作系统将腾出空间将一个就绪进程放入内存中。
      - 阻塞挂起→ 就绪挂起
        - 当它一直等待的事件发生时（状态信息可供操作系统使用）。
      - 就绪挂起 → 就绪
        - 当主内存中没有更多就绪进程时
      - 就绪→ 就绪挂起（较少可能）
        - 当没有阻塞的进程和内存时，必须释放以获得足够的性能

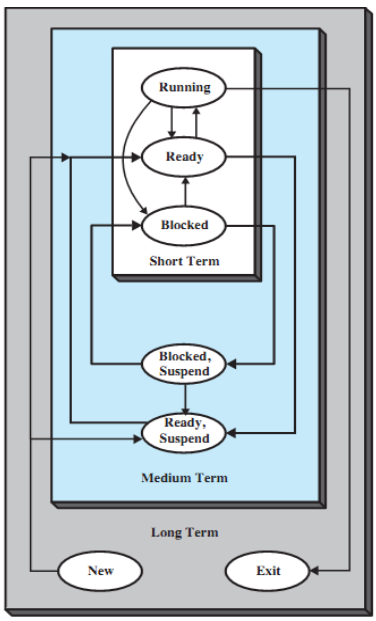
Process Scheduling

- **进程调度活动**
  - 七状态进程模型
    - 调度级别和调度活动。



Process Scheduling

- **进程调度活动**
  - 七状态进程模型
    - 三个级别的进程调度的另一种视图

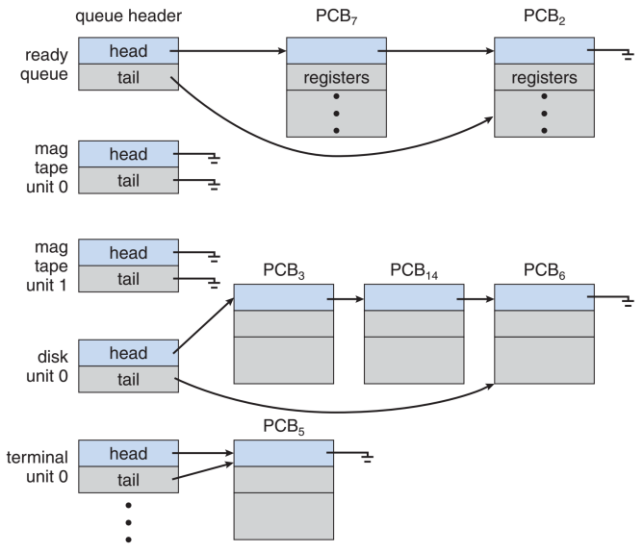


Process Scheduling

- **进程调度队列**
  - 作业队列、就绪队列和设备队列
    - 当进程进入系统时，它们被放入一个作业队列，该队列由系统中的所有进程组成。
    - 驻留在主存中且准备就绪并等待执行的进程保存在一个名为就绪队列的列表中。
      - 就绪队列通常存储为链表。表头包含指向列表中第一个和最后一个PCB的指针。每个PCB都包含一个指针字段，指向就绪队列中的下一个PCB。
    - 等待特定I/O设备的进程列表称为设备队列。每个设备都有自己的设备队列。
      - 假设一个进程向共享设备（如磁盘）发出I/O请求，但磁盘可能正忙于其他进程的I/O请求。因此，进程必须等待磁盘
      - 设备队列中记录的不是设备，而是等待特定I/O设备的进程

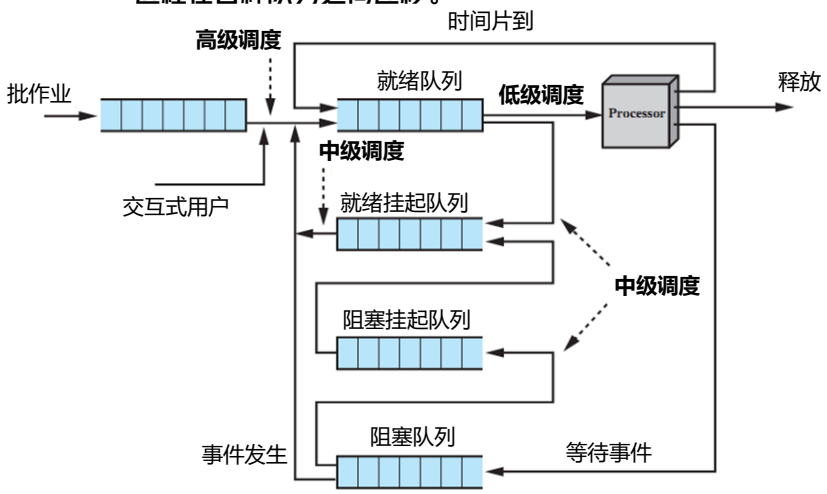
Process Scheduling

- **进程调度队列**
  - 就绪队列和各种I/O设备队列。



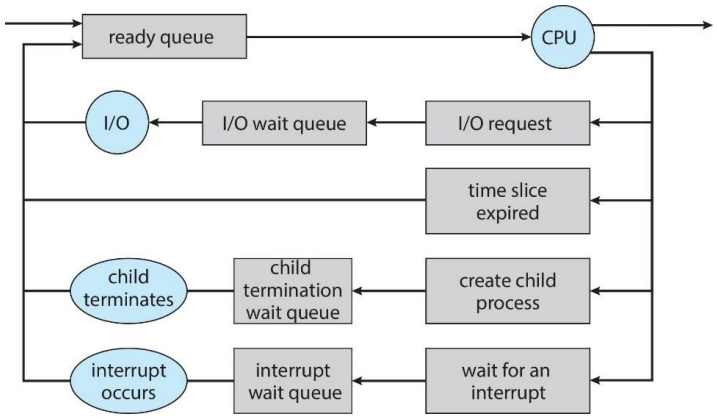
Process Scheduling

- **进程调度队列**
  - 进程调度的排队图
    - 进程在各种队列之间迁移。



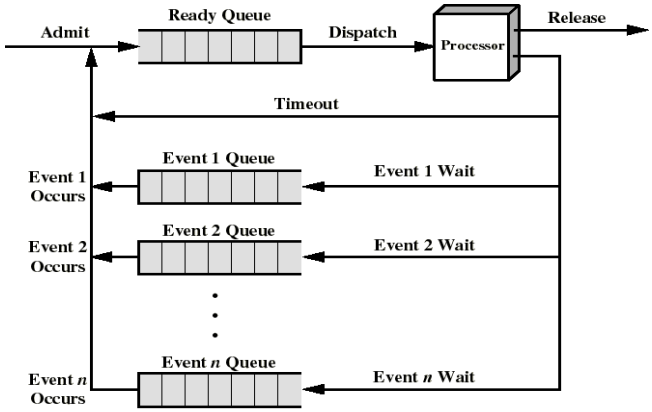
Process Scheduling

- **进程调度队列**
  - 进程调度的排队图。



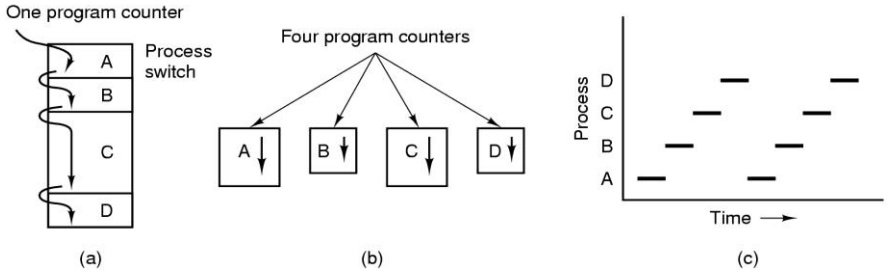
Process Scheduling

- **进程调度队列**
  - 排队规则
    - 当某个事件发生时，相应的进程被移动到就绪队列中。



Process Switching

- **进程切换**
  - 多道程序调度。



Process Switching

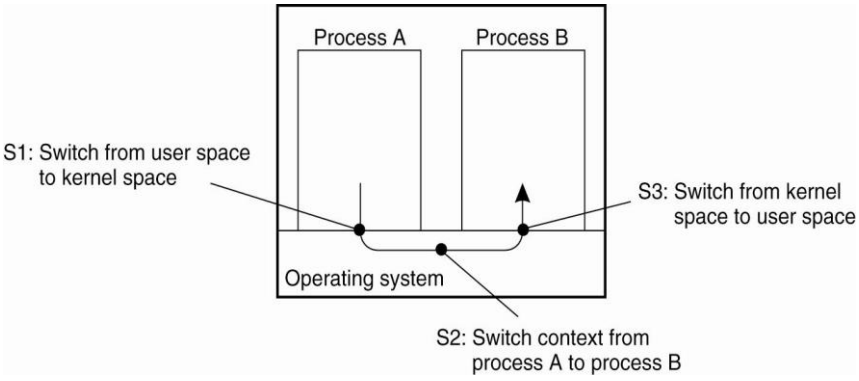
- **进程切换**
  - 进程切换发生在操作系统控制CPU时，例如，当：
    - 系统调用（Supervisor Call）
      - 程序的显式请求（例如：文件打开）
        - 这个进程可能会被阻塞
    - 陷阱
      - 上一条指令导致错误
        - 它可能导致进程移动到终止状态
    - 中断
      - 原因在当前指令的执行之外
        - 控制权被转移到中断处理程序

Process Switching

- **上下文切换**
  - 当CPU切换到另一个进程时，系统必须保存移出进程的状态，并加载移入进程的保存状态。
    - 这称为上下文切换
    - 进程的上下文在PCB中表示
  - 所需时间取决于硬件支持
  - 上下文切换时间是开销（overhead）
    - 切换时，系统在做无用功

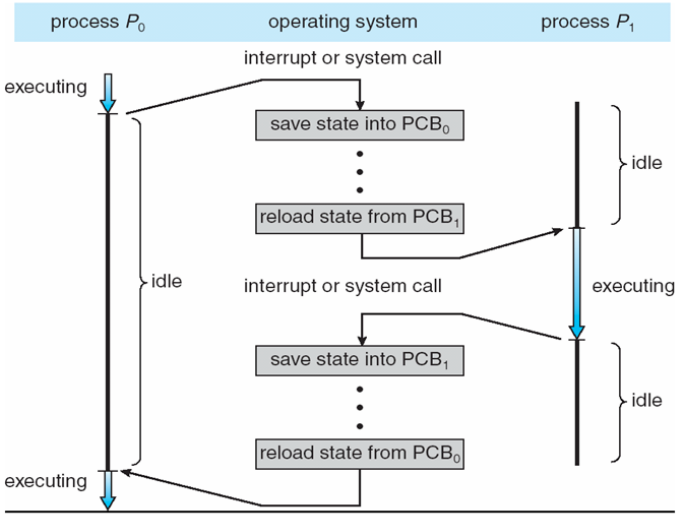
Process Switching

- 上下文切换
  - 进程之间的上下文切换。



Process Switching

- 上下文切换
  - 进程之间的上下文切换。



Process Switching

- **上下文切换**
  - 上下文切换中的步骤
    - 保存CPU上下文，包括程序计数器和其他寄存器。
    - 使用新状态和其他关联信息更新正在运行的进程的PCB。
    - 将PCB移动到适当的队列–就绪、阻塞，
    - 选择另一个进程（用于下一步执行）。
    - 更新所选进程的PCB
    - 用所选进程的CPU上下文恢复CPU的上下文。

Process Switching

- **模式切换**
  - 中断可能不会产生上下文切换。
    - 控制权可以返回到中断的程序
  - 然后，只有处理器状态信息需要保存在堆栈上
  - 这就是所谓的模式切换（Mode Switch）
    - 进入中断处理程序时，用户模式转换为内核模式。
  - 减少开销
    - 无需像上下文切换一样更新PCB。