第四章 线程和微内核技术

1. **线程-Thread**

   - **进程**：资源分配单位（存储器、文件）和cpu调度（分派）单位。又称为“任务（task）”
   - **线程**：**作为CPU调度单位**，而进程只作为其他资源分配单位
     - **只拥有必不可少的资源**

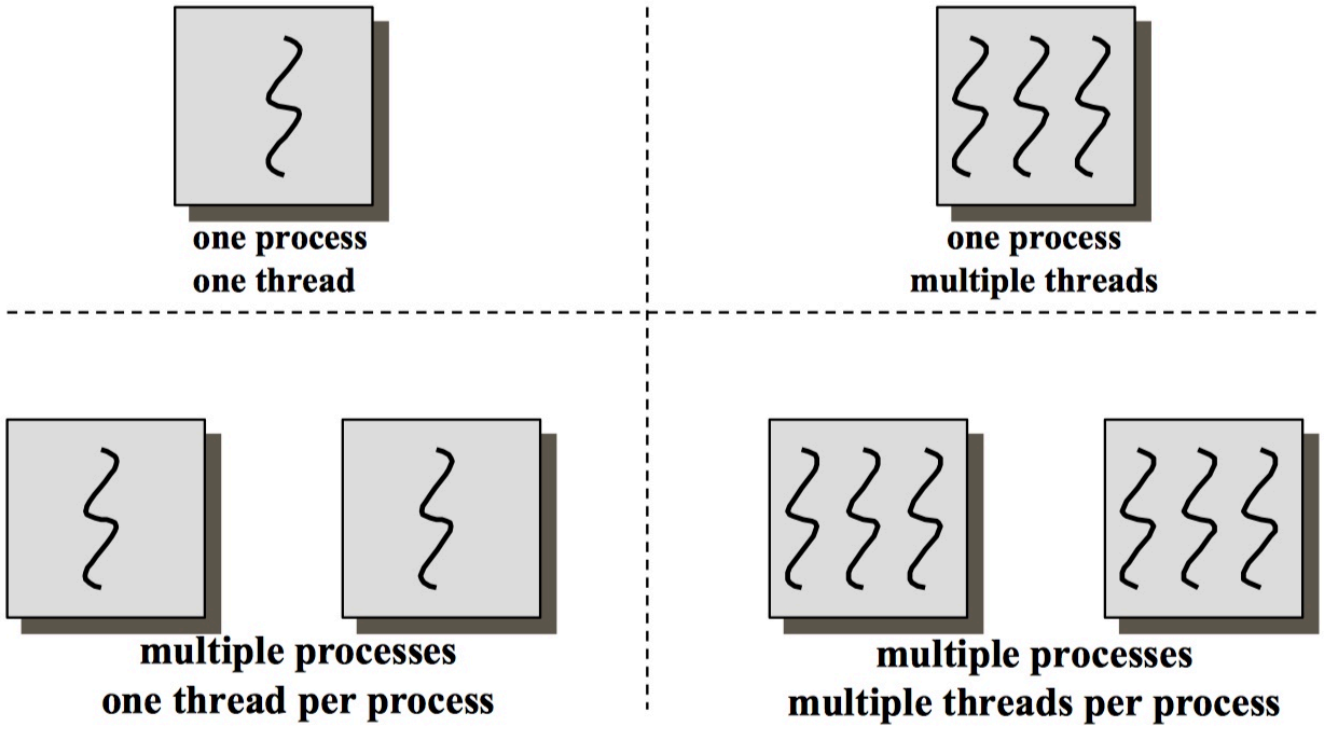
       > 如：线程状态、寄存器上下文和栈

     - 同样具有就绪、阻塞和执行三种基本状态
     - 线程控制块
   - **线程的优点**：**减小并发执行的时间和空间开销（线程的创建、退出和调度）**，因此容许在系统中建立更多的线程来提高并发程度。
     - *线程的创建时间比进程短*
     - *线程的终止时间比进程短*
     - *同进程内的线程切换时间比进程短*
     - *由于同进程内线程间共享内存和文件资源，可直接进行不通过内核的通信*

2. 进程和线程的比较

   - **地址空间和其他资源**：进程间相互独立，同一进程的各线程间共享——某进程内的线程在其他进程不可见
   - **通信**：进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要进称同步和互斥手段的辅助，以保证数据的一致性
   - **调度**：线程上下文切换比进程上下文切换要快得多

3. 进程与线程关系



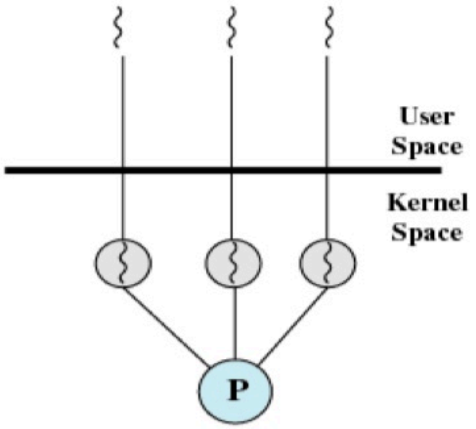
进程与线程的关系

4. **OS对线程的实现方式**

   i. *内核线程(kernel-level thread)*

   - 依赖于OS核心，由内核的内部需求进行创建和撤销，用来执行一个指定的函数。Windows NT和OS/2支持内核线程；
     - 内核维护进程和线程的上下文信息；
     - 线程切换由内核完成；
     - 一个线程发起系统调用而阻塞，不会影响其他线程的运行。
     - 时间片分配给线程，所以多线程的进程获得更多 CPU 时间。

Kernel-Level Threads

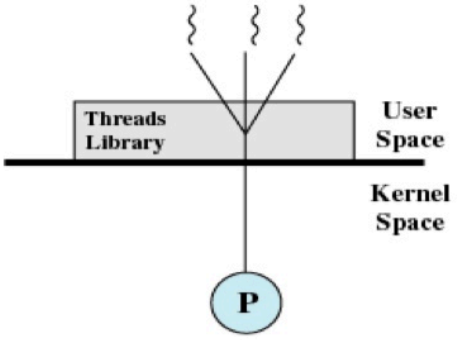


(b) Pure kernel-level

ii. *用户线程(user-level thread)*

- 不依赖于OS核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。如：数据库系统informix，图形处理Aldus PageMaker。调度由应用软件内部进行，通常采用非抢先式和更简单的规则，也无需用户态/核心态切换，所以速度特别快。 一个线程发起系统调用而阻塞，则整个进程在等待。时间片分配给进程，多线程则每个线程就慢。
  - **用户线程的维护由应用进程完成；**
  - **内核不了解用户线程的存在；**
  - **用户线程切换不需要内核特权；**
  - **用户线程调度算法可针对应用优化；**

User-Level Threads



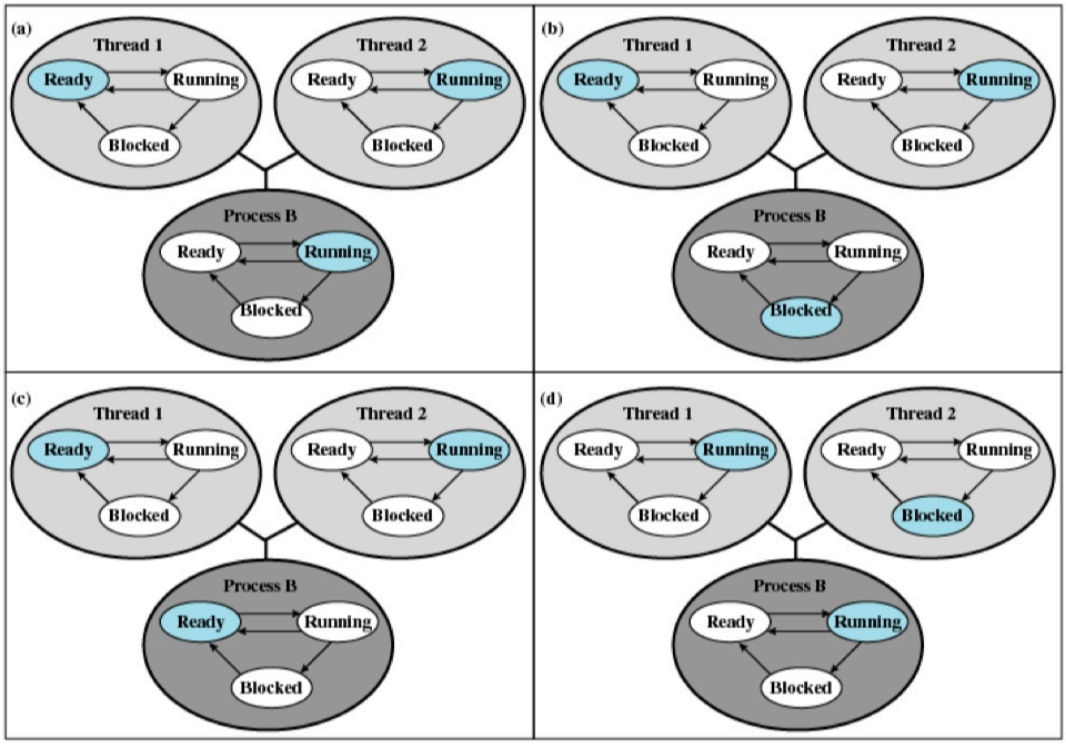
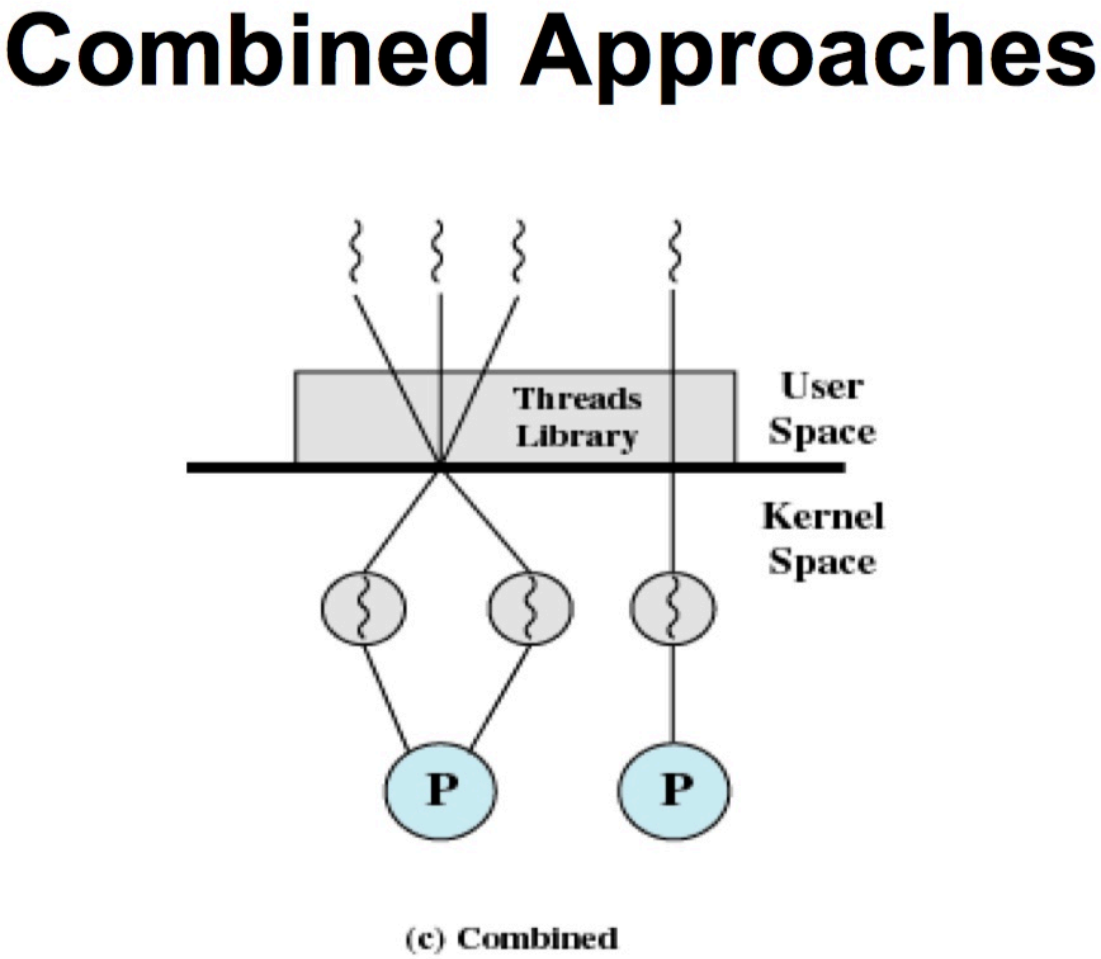


Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

iii. *混合方式*

- 比如 Solaris
- 线程在用户空间中创建

- 线程的调度和同步（scheduling and synchronization）在应用程序中进行



5. **对称多处理（SMP）** 重点

   - 所有处理器的**地位相同**
   - 系统内核可以运行在**任一**处理器上
   - 每个处理器**自主**地在统一进程/线程池内进行**调度**

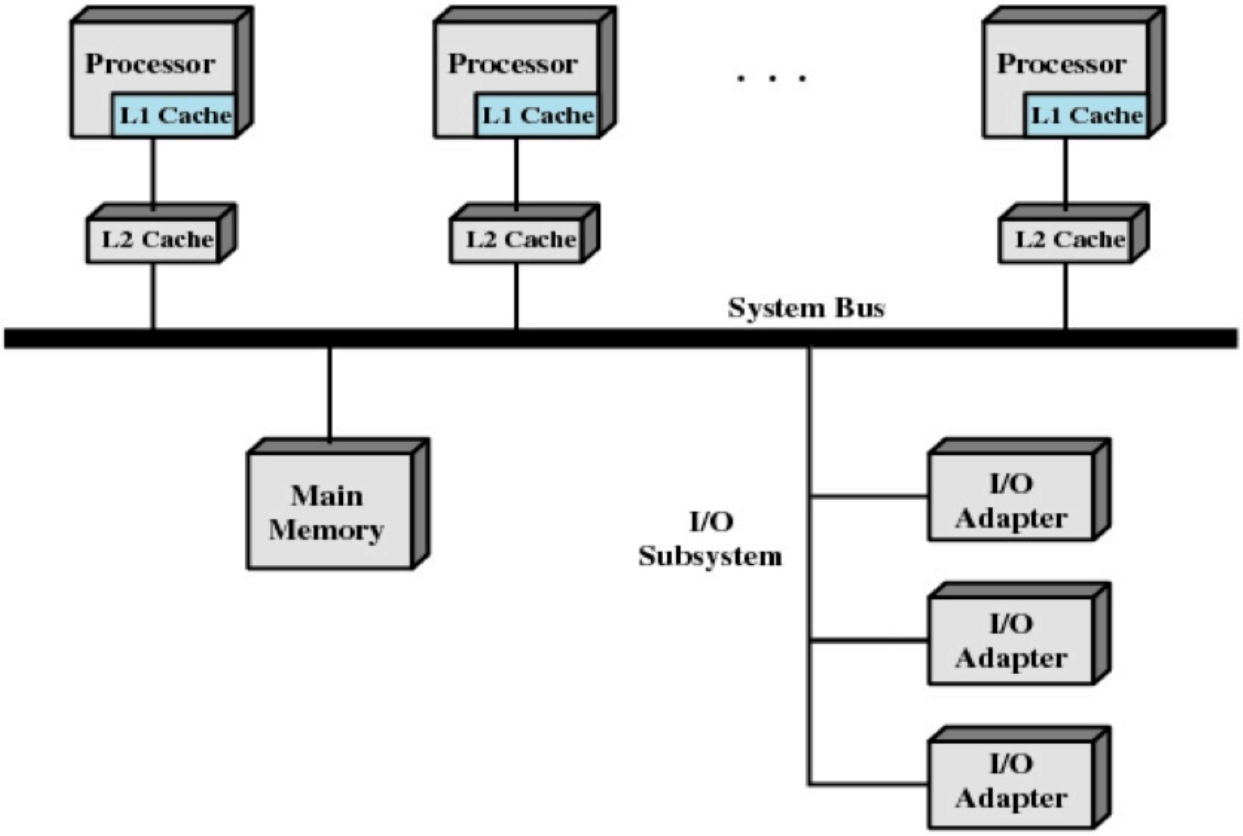


Figure 4.9 Symmetric Multiprocessor Organization

1. **SMP操作系统设计**
    - 同时并发进程或线程
    - 调度
    - 同步
    - 存储管理
    - 可靠性和容错
2. **微内核**
    - 微内核是一个小型的操作系统核心，它为模块化扩展提供了基础。
    - 定义是模糊的
    - 问题：
        - 内核多小才能称为微内核?
3. **微内核（Microkernels）**

- 只包含**基本的核心操作系统功能**
- 很多传统的操作系统服务作为**外部子系统**运行
  - 设备驱动， Device drivers
  - 文件系统， File systems
  - 虚存管理， Virtual memory manager
  - 窗口系统， Windowing system
  - 安全服务， Security services

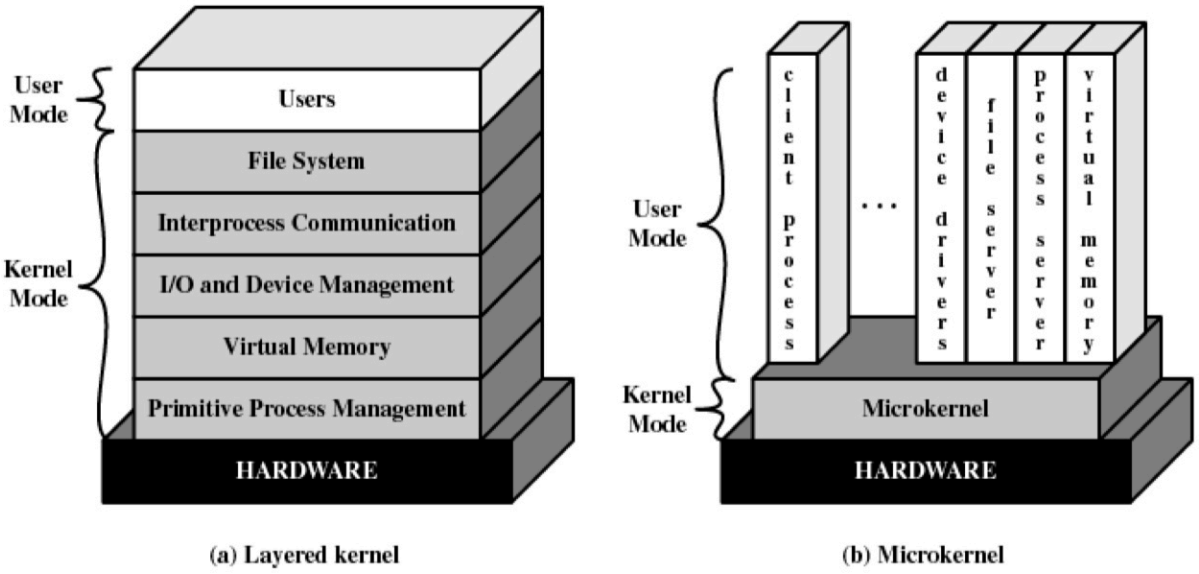


Figure 4.10 Kernel Architecture

4. 微内核的好处

- 为进程请求**提供一致接口**
  - **不区分内核级和用户级服务**
  - 所有服务通过**消息传递**的方式提供
- **可扩展性**
  - 允许添加新的服务
- **灵活性**
  - 增加新功能
  - 删减现有的功能
- **可移植性 - Portability**
  - 把内核移植到不同处理器上时，需要做的改变非常少 *macOS*
- 可靠性 - Reliability
  - 模块化设计
  - Small microkernel can be rigorously （严厉地，残酷地） tested

        - 少量的 API 提供高质量的服务
    - 有助于分布式系统支持
        - 消息传递不需要知道目标机器(processor)
        - 借助于唯一的进程和服务标识
    - 适用于面向对象的操作系统
        - 组件可以是定义了清晰接口的对象
        - 可以通过类似于搭积木的方式通过“互联”构造软件

5. 微内核设计

    - 微内核必须包括直接依赖于硬件的功能，以及那些支持服务程序和应用程序在用户模式下运行的功能。
        - 低级内存管理
        - 进程间通信
        - I/O 和中断管理

    - 低级存储器管理

        - Mapping each virtual page to a physical page frame

    - 进程间的通信

        - 全部使用消息传递，使用端口

    - I/O 和中断管理

        - 地址空间包含 I/O 端口，硬件中断当作消息处理
        - 识别中断但不处理中断，只是把中断消息传递给与该中断相关联的用户级进程