操作系统第二章习题

2.1

允许用户级进程请求操作系统服务

2.2

1. 创建和删除用户和系统进程

2. 暂停和恢复进程

3. 提供进程同步机制

4. 提供进程通信机制

5. 提供死锁处理机制

2.3

1. 跟踪当前正在使用内存的那些部分以及由谁使用

2. 当内存空间可用时，决定将哪些进程加载到内存中

3. 根据需要分配和释放内存空间

2.4

1. 空闲空间管理

2. 存储分配

3. 磁盘调度

2.5

它从用户或命令文件中读取命令并执行它们，通常将它们转换为一个或多个系统调用。

它通常不是内核的一部分，因为命令解释器可能会发生更改

2.6

在Unix系统中，需要先执行fork系统调用，然后执行exec系统调用，以启动新进程。Fork调用克隆当前正在执行的进程，而exec调用基于调用进程上的不同可执行文件覆盖新进程

2.7

系统程序可以被视为有用的系统调用包，它们为用户提供基本功能，因此用户无需自己编写程序去解决常见问题

2.8

优点：系统更容易调试和修改，因为更改只影响系统的有限部分，而不影响操作系统的所有部分。

缺点：信息只保存在需要的地方，并且只能在定义和限制的区域内访问，因此任何影响该数据的错误都必须限制在特定的模块或层中。

2.9

1. 程序执行。操作系统将文件的内容（或节）加载到内存中并开始执行。当用户级程序不被信任时无法提供。
2. I/O操作。磁盘、磁带、串行线和其他设备必须以非常低的级别进行通信。用户只需指定设备和要在其上执行的操作，而系统会将该请求转换为特定于设备或控制器的命令。不受信任的用户级程序仅访问其所有权访问的并且未被使用中的设备。
3. 文件系统操作。在文件创建、删除、分配和命名方面有许多用户不必执行的细节。磁盘空间块由文件使用，必须进行跟踪。删除文件需要删除名称文件信息并释放分配的块。还必须检查保护措施，以确保正确访问文件。用户程序既不能确保遵守保护方法，也不能在文件删除时仅分配空闲块和释放块。
4. 通信。系统间的消息传递要求将消息转换为信息包，发送给网络控制器，通过通信介质传输，并由目标系统重新组装。必须进行数据包排序和数据更正。同样，用户程序可能无法协调对网络设备的访问，或者它们可能接收到目的地为其他进程的数据包
5. 错误检测。错误检测发生在硬件和软件级别。在硬件层面，必须检查所有数据传输，以确保数据在传输过程中没有损坏。必须检查介质上的所有数据，以确保这些数据在写入介质后未发生更改。在软件层面，必须检查介质的数据一致性；例如，已分配和未分配的存储块数是否与设备上的总数匹配。在那里，错误通常与进程无关（例如，磁盘上的数据损坏），因此必须有一个全局程序（操作系统）来处理所有类型的错误。此外，通过操作系统处理错误，进程不需要包含捕获和更正系统上所有可能错误的代码

2.12

操作系统提供的一类服务是在系统中同时运行的不同进程之间实施保护。进程只允许访问与其地址空间相关联的那些内存位置。此外，不允许进程损坏与其他用户关联的文件。进程也不允许在没有操作系统干预的情况下直接访问设备。

操作系统提供的第二类服务是提供底层硬件不直接支持的新功能。虚拟内存和文件系统就是操作系统提供的两个这样的新服务事例。

2.13

1. 寄存器传递参数

2. 寄存器传递参数的起始地址

3. 参数可以由程序放置或推入堆栈，并由操作系统从堆栈中弹出

2.15

1. 创建和删除文件

2. 目录的创建和删除

3. 支持操作文件和目录的原语

4. 文件到辅助存储的映射

5. 在稳定存储介质上备份文件

2.16

优点：可以像访问文件系统中的文件一样访问每个设备。因为大多数内核都是通过文件接口来处理设备的，所以相对容易通过实现具体硬件的代码来支持抽象的文件接口来增加一个新的设备驱动器。因此，这个有利于双方用户代码的发展， 它可以以同样的方式来访问设备和文件，并且设备驱动代码可以支持一个定义明确的API。

缺点：使用相同的接口的弊端在于它可能很难捕捉在文件访问API上下文中某些设备的功能，从而导致功能损失或性能损失。

2.17

用户应该能够使用系统提供的系统调用开发新的命令解释器。命令解释器允许用户这样创建和管理进程并且确定它们的通信方式(通过管道和文件)。由于所有这些功能都可以被用户级的程序进行系统调用，因此用户可以开发出一个新的命令行解释器。

2.18

消息传递模型和共享内存模型

消息传递对于交换小数量的信息非常有用，因为不需要避免冲突。它的实现要比共享内存简单。当它在一台计算机中发生的时候，共享内存允许最大速度并且方便通信，因为它可以在内存传输。然而，这种方法在共享内存的进程之间的保护和同步方面做出了妥协。

2.19

机制和策略必须分开，来确保系统易于修改。没有两个系统的安装时相同的，所以每个安装都可能想调整操作系统以适应其需求。在机制和策略分开的情况下，策略可以随意改变，而机制则保持不变。这种安排提供了一个更灵活的系统。

2.21

1. 添加新服务不需要修改内核

2. 由于在用户模式下进行操作比在内核模式下进行的操作多，因此更安全

3. 更简单的内核设计和功能通常会带来更可靠的结果系统

用户程序和系统服务在微内核架构中通常使用进程间通信机制例如消息传递来交互。这些消息又操作系统传递。

微内核架构的主要缺点是与进程间通信相关的开销有关，并且频繁使用操作系统消息传递功能来使用户进程和系统服务可以彼此交互

2.22

1. 普通应用程序运行在用户空间，而LKM运行在内核空间。通过区分不同的运行空间，操作系统能够安全地保护操作系统中一些重要数据结构的内容不被普通应用程序所修改，达到保证操作系统正常运转的目的。
2. 普通应用程序的目标很明确，它们从头至尾都是为了完成某一项特定任务。而LKM是在内核中注册并为后续应用程序的请求提供服务的。
3. 普通应用程序可以调用并没有在其中定义的函数，但一个LKM是链接到内核上的，它所能调用的函数只有内核导出来的那些函数。
4. 普通应用程序和LKM处理错误的方式不同。当应用程序中出现错误时并不会给系统造成很大的伤害。LKM则不然，在其中出现的错误对子系统来说通常是致命的，至少对于当前正在运行的进程而言。LKM中的一个错误常常会导致整个系统崩溃。