操作系统作业 1

傅申 PB20000051

2022 年 3 月 12 日

问题 1.

系统角度

控制程序 管理用户程序的运行, 以防止计算机资源的错误或不正当使用.

资源分配器 管理计算机资源. 面对许多甚至冲突的资源请求, 操作系统考虑如何为各个程序和用户分配资源, 以便计算机系统能有效且公平地运行.

用户角度

- 操作系统设计的主要目的是用户使用方便, 次要的是性能, 不关心的是资源利用.
- 像大型机或小型机这样的共享计算机中的操作系统的设计目标是最大化资源利用率.
- 工作站等专用系统的用户拥有专用资源，但经常使用服务器的共享资源, 这类操作系统的设计需要权衡利弊, 兼顾使用方便性和资源利用率.
- 移动计算机的资源有限, 操作系统需要针对易用性和电池寿命优化.
- 嵌入式计算机等部分计算机可能几乎没有或根本没有用户界面.

问题 2.

Multiprogramming

概念 Multiprogramming 通过安排作业 (编码和数据) 使得 CPU 总是有一个执行作业, 以提高 CPU 利用率. 所有的作业首先保存在磁盘的作业池中, 其中的一部分作业被放在内存中, 通过 job scheduling 方法来选择其中一个并运行. 当作业必须等待时 (比如 I/O 中断), 操作系统将切换到另一个作业.

目的 单个程序并不能让 CPU 和 I/O 设备始终忙碌, 而用户通常有多个程序, 通过 multiprogramming 可以提高资源利用率

Multitasking

概念 Multitasking 是指 CPU 切换作业的频率很高, 用户可以在程序运行时与其交互, 这样也可以允许许多用户共享一台计算机.

目的 Multitasking 是 multiprogramming 的自然延伸.

问题 3.

存储层次

片上缓存 寄存器和高速缓存, 容量极小, 速度很快.

内存 随机访问, 容量少, 具有易失性, 直接被 CPU 访问.

外存 磁盘或硬盘, 提供大容量不易失的存储能力.

外部存储设备 如磁带和光盘

缓存思想 缓存是计算机系统的一条重要原理. 信息通常保存在一个存储系统中, 使用时, 它会被临时拷贝到更快的存储系统中, 即高速缓存. 当需要特定信息时, 首先检查它是否处于高速缓存中, 如果是, 信息直接在高速缓存中被使用, 否则使用位于源地的信息并将其拷贝到高速缓存中.

问题 4.

系统调用是操作系统提供服务的编程接口, 为用户程序提供了要求操作系统执行任务的途径. 系统每秒执行成千上万的系统调用.

通常, 我们通过高级语言的 API 来访问系统调用而不是直接使用系统调用. API 是为方便应用程序员而规定的一组函数. 系统调用接口截取 API 函数的调用, 并调用操作系统中所需要的系统调用.

问题 5.

工作机制 Dual-mode 至少需要两种单独运行模式: 用户模式和内核模式. 计算机硬件可以通过一个模式位来表示和区分当前模式. 当计算机系统执行用户应用时, 系统处于用户模式. 当应用通过系统调用请求操作系统服务时, 系统必须从用户模式切换到内核模式. 系统调用返回结果后, 又切换回用户模式.

原因 Dual-mode 可以让操作系统保护自己以及其他系统组件. 当尝试在用户模式执行特权指令时, 硬件不会执行该指令, 而是认为该指令非法, 并通知操作系统.

问题 6.

提供用户功能

程序执行 加载程序到内存并加以运行. 程序应该能结束运行, 包括正常或不正常 (并给出错误).

I/O 操作 为了效率和保护, 用户不应该直接控制 I/O 设备. 操作系统必须提供手段以便执行 I/O.

文件系统操作 操作系统提供多种文件系统.

通信 进程之间交换信息.

错误检测 操作系统需要不断检测错误和更正错误.

用户界面 CLI, GUI, 批处理

确保系统高效

资源分配 操作系统管理许多不同类型的资源, 分配到各个用户 / 作业.

记账 记录用户使用资源的类型和数量.

保护与安全 操作系统需要控制计算机的使用, 保证并发执行的独立进程不会互相干扰, 以及系统安全不受外界侵犯.

问题 7.

单片结构 在一层里面包含了大量功能. 缺点是难以实现与设计, 优点是系统调用接口和内核通信的开销非常小.

层次化结构 将操作系统分成若干层 (最低为硬件, 最高为用户接口), 操作系统层采用抽象对象以包括数据和操作. 优点是简化了构造和调试, 并且隐藏了高层的数据结构, 操作和硬件. 缺点是难以定义各层, 并且效率稍差.

模块化结构 采用可加载的内核模块, 内核提供核心服务, 其他额外服务通过模块链入. 优点是更加灵活和高效.

微内核结构 从内核中删除所有不必要的部件, 将它们当作系统级与用户级的程序来实现. 优点是便于拓展, 便于移植, 并且更加可靠与安全. 缺点是性能会受损.

问题 8.

机制决定如何做, 策略决定做什么. 比如定时器是一种保护 CPU 的机制, 但是为某个特定用户应该将定时器设置成多长时间就是一个策略问题. 机制是针对某一问题一种通用的方法, 而对于具体情况如何去实施这一方法就是策略问题.

该设计的好处是能增大灵活性, 在实施时只需要改变策略而不用改变机制.