OPERATING SYSTEMS

Structures of Operating Systems

Operating Systems

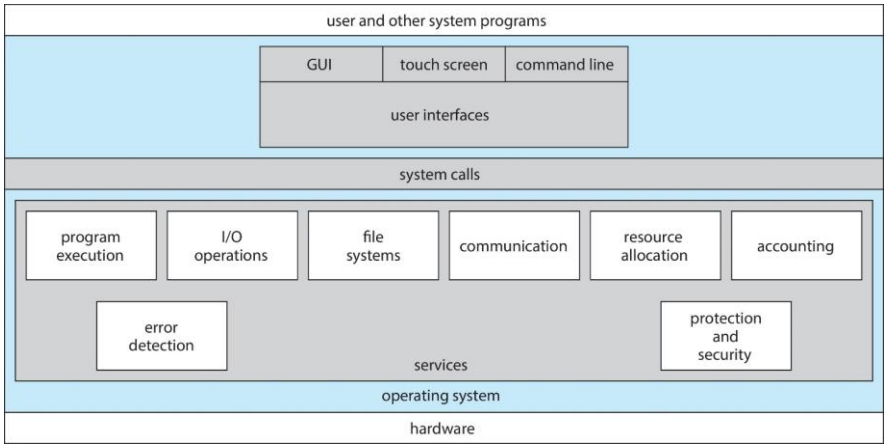
郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- **目录**
  - **操作系统服务**
  - **通用操作系统组件**
  - **系统调用和API**
  - **系统程序**
  - **操作系统设计与实现**
  - 操作系统的结构/组织/布局
    - 单片（一个非结构化程序）
    - 分层
    - 微核
    - 虚拟机

Operating System Services

- 操作系统服务
  - 操作系统服务视图
    - 提供这些操作系统服务是为了方便程序员，使编程任务更容易
    - 面向用户的视图。



Operating System Services

- 操作系统服务
  - 用户界面（UI）
    - 命令行界面（CLI）
      - 使用文本命令和输入的方法
      - 例如，键盘打字。
    - 批处理接口
      - 命令与控制这些命令的指令被输入到文件中
      - 解释和执行这些文件
    - 图形用户界面（GUI）
      - 该界面是一个窗口系统，带有指向设备，用于指示I/O、从菜单中选择、确认选中，以及输入文本的键盘。

Operating System Services

- 操作系统服务
  - 系统调用
    - 程序执行
      - 操作系统能够将程序加载到内存中，运行它，正常或异常地结束执行
    - I/O操作
      - 不允许用户程序直接执行I/O操作。操作系统必须提供一些执行I/O的方法，这可能涉及文件或I/O设备。
    - 文件系统
      - 读取、写入、创建和删除文件和目录的程序功能
    - 通信
      - 进程可以在同一台计算机上或通过网络在计算机之间交换信息——通过共享内存或消息传递实现。
        - 进程间通信（IPC）

Operating System Services

- 操作系统服务
  - 系统调用
    - 错误检测
      - 通过检测CPU和内存硬件、I/O设备或用户程序中的错误来确保正确的计算
    - 资源分配
      - 当多个用户或多个作业同时运行时，必须为每个用户或作业分配资源。
    - 会计
      - 跟踪哪些用户使用了多少以及哪些类型的计算机资源
    - 保护和安全
      - 并发进程不应相互干扰，也不应干扰操作系统进程。
      - 制定预防措施以防止误用或恶意活动。

Common Operating System Components

- 通用操作系统组件
  - 操作系统组件是为提供系统服务而设计和实现的。
  - 面向系统的视图
    - 过程管理
    - 主存管理
    - 文件管理
    - 大容量存储管理
    - I/O管理

    资源管理（上一节已介绍）
    - 网络
    - 命令行解释器
    - GUI
    - 保护和安全
    - 错误检测与响应
    - 会计

Common Operating System Components

- 网络
  - 系统中的处理器通过通信网络连接。
    - 通信使用协议进行。
    - 网络化/分布式系统为用户提供对各种系统资源的访问。
  - 访问共享资源允许：
    - 计算加速
    - 提高数据可用性
    - 增强可靠性

Common Operating System Components

- 命令行解释器
  - 读取和执行给操作系统的命令的程序
  - 命令行解释器示例
    - Command.com（MS-DOS）
    - Shell（UNIX）
  - 在窗口系统中，界面基于鼠标和菜单
    - WIMP（窗口、图标、菜单和定点设备）

Common Operating System Components

- 图形用户界面（GUI）
  - 用户友好的桌面界面：
    - 通常是鼠标、键盘和显示器。
    - 图标代表文件、程序、动作等。
    - 界面中对象上的各种鼠标按钮会导致各种操作。
      - 提供信息、选项、执行功能、打开文件夹
    - 发明于Xerox PARC施乐帕克研究中心
  - 系统现在包括CLI和GUI界面：
    - Microsoft Windows是带有CLI “command” shell的GUI。
    - 苹果macOS X使用“Aqua”图形用户界面，内核为UNIX，也有shell可用。
    - Solaris是具有可选GUI（Java桌面，KDE）的CLI。
  - Linux内核只是CLI，没有GUI。
    - Gnome（GNU网络对象模型环境）和KDE（King桌面环境）是在X协议下开发的图形环境。

Common Operating System Components

- 保护和安全
  - 保护
    - 控制进程或用户对系统和用户资源的访问的机制
  - 安全
    - 系统防御内部和外部攻击，范围广泛，包括：
      - 拒绝服务DoS
      - 蠕虫Warms
      - 病毒Viruses
      - 身份盗窃
      - 盗窃服务
      - …

Common Operating System Components

- 保护和安全
  - 系统通常首先区分用户，以确定谁可以做什么：
    - 用户ID包括名称和相关编号，每个用户一个。
    - 用户ID与该用户的所有文件、进程关联，以确定访问控制。
    - 组ID允许定义和管理一组用户，也与每个进程、文件关联。
    - 权限升级允许用户更改为具有更多权限的有效ID。

Common Operating System Components

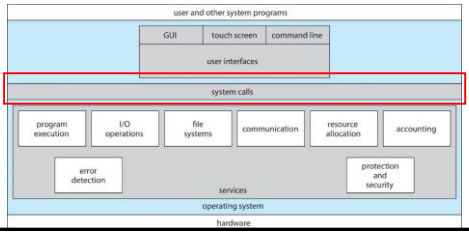
- 错误检测与响应
  - 错误检测
    - 内部和外部硬件错误
      - 内存错误
      - 设备故障
    - 软件错误
      - 算术溢出
      - 访问禁止的内存位置
  - 错误响应
    - 只需向应用程序报告错误
    - 请重试该操作
    - 中止应用程序

Common Operating System Components

- 会计
  - 会计跟踪并记录哪些用户使用了多少以及使用了哪些类型的计算机资源。
    - 收集有关资源使用情况的统计数据
    - 监控性能（例如，响应时间）
    - 用于系统参数调整以提高性能
    - 用于预测未来的改善
    - 用于向用户计费（在多用户系统上）

System Calls and APIs

- 系统调用和API
  - 系统调用为操作系统提供的服务提供编程接口。
    - 通常用高级语言（C/C++）编写。
    - 主要由程序通过高级应用程序接口（API）访问，而不是通过直接的系统调用。
  - 三种最常见的API：
    - 适用于Windows的Win32 API
    - 用于基于POSIX的系统的POSIX API
      - 包括几乎所有版本的UNIX、Linux和mac OS X。
      - POSIX： Portable Operating System Interface
    - Java虚拟机（JVM）的Java API。



System Calls and APIs

- 系统调用和API
  - 系统调用的示例

source file → 复制到 → destination file

Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

System Calls and APIs

- 系统调用和API
  - 标准API示例
    - 考虑Win32 API中的ReadFile()—读取文件的函数

```
BOOL ReadFile(
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD  nBytesToRead,
    LPDWORD lpBytesRead,
    LPOVERLAPPED lpOverlapped
);
```

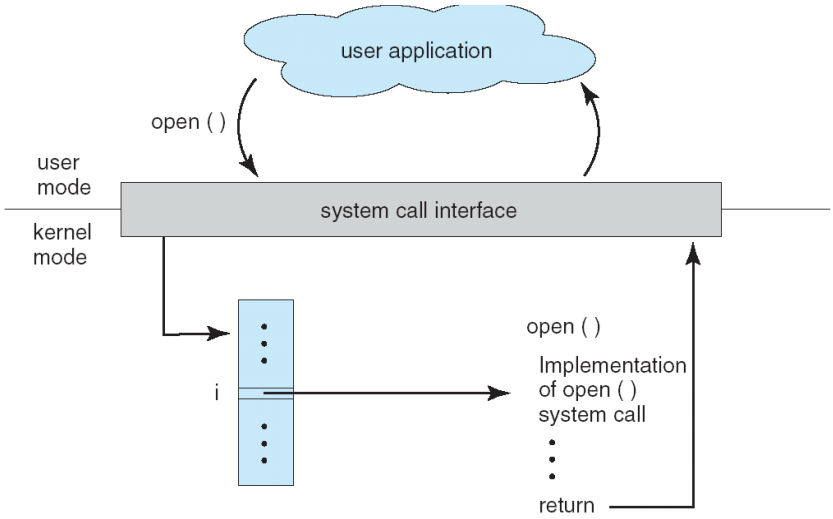
    - 传递给ReadFile()的参数的说明
      - HANDLE `hFile` 要读取的文件
      - LPVOID `lpBuffer` 将数据读入和写入的缓冲区
      - DWORD `nBytesToRead` 要读入缓冲区的字节数
      - LPDWORD `lpBytesRead` 上次读取期间读取的字节数
      - LPOVERLAPPED `lpOverlapped` 指示是否正在使用重叠I/O

System Calls and APIs

- 系统调用和API
  - 系统调用实现
    - 通常，一个号码与每个系统调用相关联。
      - 系统调用接口根据系统调用号维护索引表。
    - 系统调用接口在内核中调用预期的系统调用，并返回系统调用的状态和任何返回值。
    - 调用者不需要知道系统调用是如何实现的：
      - 只需遵守API语法并了解操作系统返回的结果意味着什么
      - API对程序员隐藏了操作系统接口的细节，因为它是由运行时支持库（编译器附带的库中内置的一组函数）管理的。

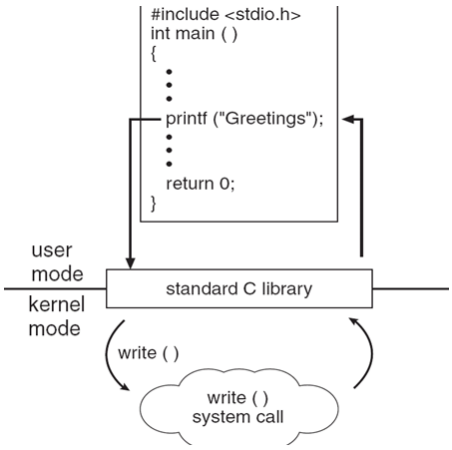
System Calls and APIs

- 系统调用和API
  - API、系统调用和操作系统之间的关系。



System Calls and APIs

- 系统调用和API
  - 标准C库
    - 示例：调用标准IO库的printf()的C程序，将在内核调用write()系统调用

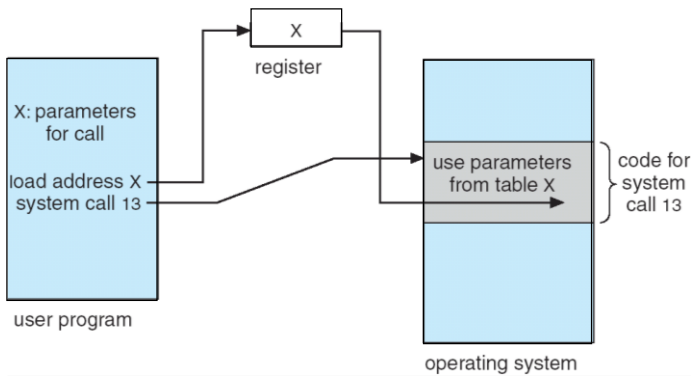


System Calls and APIs

- 系统调用和API
  - 系统调用参数传递
    - 除系统调用号外，通常需要更多的参数，但参数的确切类型和数量因操作系统和调用而异。
    - 用于向操作系统传递参数的三种通用方法：
      - 最简单地在寄存器中传递参数。
        - 在某些情况下，参数可能比寄存器多。
      - 存储在内存块或表中的参数，以及作为参数在寄存器中传递的块地址。
        - Linux和Solaris采用这种方法。
      - 参数由程序放置或压入堆栈上，并由操作系统弹出堆栈。
    - 块和堆栈方法不限制所传递参数的数量或长度。

System Calls and APIs

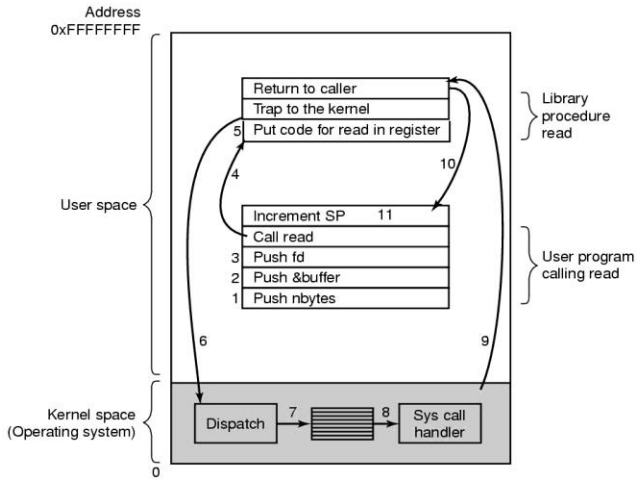
- 系统调用和API
  - 系统调用参数传递
    - 内存表传参



系统调用直接从约定的寄存器里X内存表地址读取参数值！

System Calls and APIs

- 系统调用和API
  - 系统调用参数传递
    - 通过堆栈传递：系统调用read(fd, buffer, nbytes)



System Calls and APIs

- 系统调用和API
  - POSIX系统调用示例
    - POSIX: UNIX的可移植操作系统接口，IEEE 1003.X 1992-1998

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

System Calls and APIs

- 系统调用和API
  - POSIX系统调用示例
    - POSIX: UNIX的可移植操作系统接口，IEEE 1003.X 1992-1998

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

System Calls and APIs

- 系统调用和API
  - Win32 API调用大致对应于UNIX调用

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

System Programs

- 系统程序
  - 系统程序为程序开发和执行提供了方便的环境。其中一些只是系统调用的用户接口；其他的则要复杂得多。
  - 大多数用户对操作系统的看法是由系统程序定义的，而不是实际的系统调用。
  - 系统程序提供：
    - 文件和目录管理/修改
    - 状态信息
    - 编程语言支持
    - 程序加载和执行
    - 通信
    - 应用程序

System Programs

- 系统程序
  - 文件和目录管理
    - 创造
    - 删除
    - 复制
    - 改名
    - 打印
    - 转存（Dump）
    - 列表
  - 文件修改
    - 用于创建和修改文件的文本编辑器。
    - 用于搜索文件内容或执行文本转换的特殊命令。

System Programs

- 系统程序
  - 状态信息
    - 一些系统程序要求系统提供信息
      - 日期
      - 时间
      - 空闲内存
      - 可用磁盘空间
      - 用户数
    - 其他的则提供详细的性能、日志记录和调试信息。
    - 通常，这些系统程序格式化并将所需的状态信息打印到终端或其他输出设备。
    - 一些系统实现了用于存储和检索配置信息的注册表。

System Programs

- 系统程序
  - 编程语言支持
    - 编译程序
    - 汇编程序
    - 调试器
    - 解释程序
  - 程序加载和执行
    - 绝对装载机
    - 可移动装载机
    - 链接编辑器
    - 覆盖装载机
    - 调试系统

System Programs

- 系统程序
  - 通信
    - 为以下对象之间创建虚拟连接提供机制
      - 进程
      - 使用者
      - 计算机系统
    - 网络通信允许用户向彼此的屏幕发送消息、浏览网页、发送电子邮件、远程登录、将文件从一台机器传输到另一台机器
  - 应用程序示例
    - 数据库系统
    - 编译程序
    - 网络浏览器
    - 文字处理器
    - 文本格式化程序
    - 电子表格
    - 绘图和统计分析软件包
    - 游戏

Operating System Design and Implementation

- 操作系统设计与实现
  - 操作系统的设计和实现不是“可解决的”，但一些方法已被证明是成功的。
    - 不同操作系统的内部结构可能差异很大。
    - 受硬件选择、系统类型的影响。
    - 从定义目标和规范开始。
  - 用户目标和系统目标
    - 用户目标
      - 操作系统应该使用方便、易学、可靠、安全和快速。
    - 系统目标
      - 操作系统应该易于设计、实现和维护，以及灵活、可靠、无错误和高效。

- 操作系统设计与实现
  - 机制和策略的分离
    - 策略与机制的分离是一个非常重要的原则，如果策略决定以后要改变，它允许最大限度的灵活性。
      - 策略：将采取什么措施？
      - 机制：如何做？
    - 实例
      - 存在一个抽象优先级队列，我们需要支持以下机制：
        - 在开头插入/删除项目。
        - 在末尾插入/删除项目。
        - 知道队列的长度。
      - 队列的主体/代码可以用不同的方式实现。
      - 策略可以是FIFO、LIFO–应由队列用户决定。

- 操作系统设计与实现
  - 系统实现
    - 传统上是用汇编语言编写的，操作系统的大多数部分现在都可以用高级语言编写
    - 用高级语言编写的代码：
      - 写得更快
      - 更紧凑
      - 易于理解和调试
    - 如果操作系统是用高级语言编写的，那么移植到其他硬件就容易得多。

- 操作系统设计与实现
  - 开源操作系统
    - 提供源代码格式，而不仅仅是封闭源代码的二进制执行文件。
    - 反对拷贝保护和数字版权管理（DRM）运动。
    - 由自由软件基金会（FSF）发起，它拥有“公共非盈利版权”GNU公共许可证（GPL）。
    - 例如，GNU/Linux、BSD UNIX等

- 操作系统设计与实现
  - 操作系统调试
    - 调试是发现并修复错误或bug。
    - 布莱恩·克尼汉定律（*Brian Kernighan*）
      - “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”
      - *Brian W. Kernighan*和*Dennis M. Ritchie*在1978年发明了C语言
    - 操作系统生成包含错误信息的日志文件。
      - 应用程序故障会生成核心转储文件，捕获进程内存。
      - 操作系统故障可能会生成包含内核内存的崩溃转储文件。
    - Solaris、FreeBSD、Mac OS X中的DTrace工具
      - 执行代码时会触发探测，捕获状态数据并将其发送给这些探测的使用者。

- 操作系统设计与实现
  - 操作系统生成和引导
    - 操作系统设计为在一类机器的任意一台上运行；必须为每台特定计算机配置操作系统
      - SYSGEN程序获取硬件系统特定配置的有关信息
    - 引导
      - 操作系统必须对硬件可用，使硬件可在加载内核时启动它
    - 引导程序
      - 引导程序是存储在ROM或EPROM（固件）中的一小段代码
      - 通电或重新启动时，引导程序从固定内存位置开始执行。
        - 有时，在固定位置的引导块加载引导加载程序有两步过程
      - 引导程序定位内核，将其加载到内存中，然后开始执行。
        - 初始化系统的方方面面