操作系统 5

jask

09/15/2024

内存

主要问题：如何将有限的内存分配给多个程序使用?

直接做法是：将内存按地址分配给程序，加入 A 需要 10mb, B 需要 100mb, 就把内存 0-10mb 给 A, 10-110mb 给 B。

这样做的问题很多

地址空间不隔离

所有程序都访问物理内存，内存空间隔离，很容易破坏别的程序。

内存使用效率低

在程序调度的过程中，每一个程序都会产生大量的换入换出，导致效率低下。

程序运行的地址不确定

程序每一次装入运行时，都要重新分配一块内存，这导致程序编写困难，因为编写时很多访问数据和指令跳转时的目标地址是固定的。这设计重定向问题。

解决方法 1:

分段

将一个程序需要的内存空间大小的虚拟空间映射到某个地址空间，映射地址有软件设置，比如操作系统设置映射函数，实际转换由硬件完成。

解决了 1,3 问题，但是 2 没解决。

分页

把地址空间人为分成大小固定的页，每一页的大小由操作系统决定。

虚拟空间的页叫虚拟页，物理内存中的页叫内存页，磁盘中的页叫磁盘页。

虚拟空间的有些页被映射到同一个物理页，就实现了内存共享。

同时实现了保护，通过每个页设置权限属性就可以决定谁可以修改谁可以访问。

页映射下，CPU 发出的是 virtual address, 要经过 mmu 转换为 physical address。

线程

访问权限

私有：栈，线程局部变量，寄存器

共享：全局变量，堆上的数据，函数里的静态变量，程序代码，打开的文件（描述符等）

轮转调度决定了线程之间交错执行的特点，优先级调度决定了线程按照什么顺序轮流执行。

频繁等待的线程成为 IO 密集型线程。占用大量 CPU 时间的叫 CPU 密集型线程。

可抢占线程与不可抢占

线程在时间片用尽之后强制被剥夺执行的权利而进入就绪状态，叫做抢占。

Linux 多线程

Linux 将所有执行实体（不管是进程还是线程）都叫做任务，每一个任务都类似于一个单线程的进程，具有内存空间、执行实体、文件资源，但是 Linux 下的不同任务可以选择共享内存空间，所以共享同一个内存空间的多个任务构成了进程。

fork 复制当前进程，但是并不复制原任务的内存空间，而是和原任务一起共享一个 Copy On Write 的内存空间。fork + exec 使用产生新任务，fork 产生任务，exec 执行新的可执行文件。

创建新线程使用的是 clone。

可重入与线程安全

一个函数被重入，表示这个函数没有执行完成，由于外部因素或者内部调用，又一次进入函数执行。

两种情况：

1. 多个线程同时执行这个函数。

2. 函数自身调用自身。

一个函数成为可重入的，表明该函数被重入之后不会产生任何不良后果。

可重入函数的特点：

1. 不使用任何（局部）静态或全局的非 const 变量

2. 不返回任何（局部）静态或全局的非 const 变量的指针。

3. 仅依赖于调用方提供的参数。

4. 不依赖任何单个资源的锁。

5. 不调用任何不可重入的函数。

链接

链接包括：地址与空间分配，符号决议和重定位。

函数级别链接

gcc 提供了-ffunction-sections 和-fdata-sections 的选项可用于将每个函数/变量分别保持到独立的段中。

msvc 提供了函数级别链接。

动态链接

静态链接对于内存和磁盘的空间浪费非常严重。

动态链接的好处：

1. 减少了物理页面的换入患处

2. 增加了 cpu 缓存的命中率，因为不同进程间的数据和指令访问都集中在同一个共享模块上。

3. 程序在运行时可以动态地选择加载各种程序模块，这个优点后来被人们用来制作程序的插件。

地址无关代码

-fPIC 可以使指令在多个进程之间共享

延迟绑定

ELF 采取的是函数第一次被用到时才进行绑定，没有用到则不进行绑定。

ELF 采用 PLT(Procedure Linkable Table) 来实现。

装载（load）

覆盖装入

编写程序时手工将程序分割成若干块，编写一个小的辅助代码来管理这些模块何时应该驻留内存而何时被替换掉，这个代码就是所谓的覆盖管理器。

往往用于内存极度受限的地方。

页映射

参考现代操作系统的存储管理器。

进程的建立

1. 创建一个独立的虚拟地址空间

并不是创建空间，而是创建映射函数所需要的相应的数据结构。

2. 读取可执行文件头，并且建立虚拟空间与可执行文件的映射关系

建立虚拟空间与可执行文件的映射。这种映射关系只是保存在操作系统内部的一个数据结构，Linux 将进程虚拟空间中的一个段叫做虚拟内存区域（VMA, virtual memory area），Windows 中将这个叫做虚拟段（Virtual Section）

3. 将 CPU 指令寄存器设置成可执行文件入口

操作系统通过设置 CPU 的指令寄存器将控制权转交给进程，由此进程开始执行，