操作系统

陶

2024 年 3 月 1 日

目录

1 操作系统的介绍

1.1 当程序运行时发生了什么

执行指令：处理器从memory fetch instructions，解码并且执行。（the basic of Von Neumann model of computing）

1.2 操作系统的各种称呼（目标都是为了更好用）

虚拟机：将处理器，内存，磁盘等虚拟化标准库：提供了各种api方便系统调用资源管理器：管理处理器，内存，磁盘等资源

1.3 三个主题

1.3.1 Persistence

内存断电后清空，但用户不希望丢掉数据：使用file system管理文件，以及发展出I/O设备（HD SSD）长期储存

1.3.2 Concurrency

1.3.3 Virtualization

1.4 设备驱动

操作系统下载驱动后明白如何使用该设备

1.5 写的协议

1.5.1 journaling

1.5.2 copy-on-write

1.6 如何构建系统

1.7 目的

1.7.1 performance

1.7.2 protection

isolation

1.7.3 reliability

1.7.4 energy-efficiency

1.7.5 security

1.7.6 mobility

1.8 Abstraction

2 The Abstraction: The Process—-running program

虚拟化CPU

2.1 policies

algorithms for making some kind of decision within the OS

2.2 进程的组成

2.2.1 进程映像

程序，数据集合，栈，PCB（process control block）构成进程在系统中存在和活动的实体

2.2.2 PCB：进程描述块，进程存在的唯一标志

进程名：标识对应进程的唯一标识符或数字

特征信息：是系统进程还是用户进程，进程实体是否常驻内存

进程状态信息：运行态/就绪态/阻塞态

调度优先权：进程获取CPU的优先级别

通信信息：反映该进程与哪些进程有什么样的通信关系，如等待哪个进程的信号。

现场保护区：对应进程由于某个原因放弃使用CPU时，需要把它的一部分与运行环境有关的信息保存起来，以便在重新获得CPU后可以恢复正常运行。通常被保护的信息有程序计数器，程序状态字，工作寄存器的内容等。

资源需求，分配和控制方面的信息：如进程所需或占用的I/O设备，磁盘空间，数据区等

进程实体信息：指出该进程的程序和数据的存储情况，在内存或外存的地址，大小等。

族系关系：反映父子进程的隶属关系

其他信息：文件信息，工作单元

2.3 进程队列

为了对进程进行有效管理，将各进程的PCB组织起来形成进程队列。

2.3.1 线性方式

数组。操作系统预先确定整个系统中同时存在的进程最大数目，静态分配空间。

2.3.2 链接方式

按照进程的不同状态，将其分别放在不同链表中。阻塞队列可以有多个对应不同的阻塞原因。实际系统中就绪队列按照进程优先级分成多个队列，有同一优先级的排在同一个队列上。

2.3.3 索引方式

根据进程不同的状态建立索引表，索引表条目存放PCB地址，将索引表的起始地址放在专用的指针单元中

2.4 进程管理

2.4.1 创建进程 fork()

1.从系统的PCB表中找出一个空闲的PCB项，并指定唯一的进程标识号（Process identifier,PID），用作进程内部名

2.根据调用者提供的所需内存大小，为新进程分配必要的内存空间，用于存放其程序，数据和工作区。有两种可能：（1）子进程复制父进程的地址空间（2）将新的程序装入子进程的地址空间

3.根据调用者提供的参数，将新进程的PCB初始化。参数包括新进程名（外部标识符），父进程标识符，处理机初始状态，进程优先级，本进程的开始地址等。一般将新进程状态设置为就绪态。

4.一个新进程派生新进程后，有两种可能的执行方式：（1）异步方式：父进程继续运行，子进程也可以被调度运行（2）同步方式：父进程睡眠，等待某个或全部子进程终止，然后继续运行

父进程调用fork创建子进程时，将自己的地址空间制作一个副本，其中包括User结构，正文段，数据段，用户栈和系统栈，使得父进程很容易和子进程通信。两个进程都可以继续执行fork后的指令。当fork的返回值（子进程PID）不等于0，表示父进程在执行，当fork 返回值为0时表示子进程在执行

此时子进程中的程序为父进程调用fork后的指令。

subsubsection终止进程 primitive

1.从系统的PCB表中找到指定进程的PCB。若它处于运行态，则立即终止该程序的运行。

2.回收该进程所占用的全部资源

3.如果该程序有子孙进程，则要终止其所有子孙进程并且回收它们的全部资源

4.释放原本进程的PCB，并将其从原本队列中摘下

2.4.2 阻塞进程 sleep

不满足继续的条件，主动调用sleep阻塞自己。

1.立即停止当前进程的执行

2.将该进程的CPU现场送到该进程的PCB现场保护区中保存

3.将该进程中PCB的现行状态由运行态改为阻塞态，并将其插入有相同事件的阻塞队列中。

4.转到进程调度程序，重新从就绪队列中挑选一个合适的进程运行

2.4.3 唤醒进程 wakeup

当阻塞进程所等待的事件出现时（等待数据到达或I/O操作完成）

1.将被阻塞进程从阻塞队列摘下

2.将现行状态改为就绪态，将该进程插入就绪队列

3.如果被唤醒的进程比正在运行的进程有更高的优先级，则重新调度标志

sleep 为自己主动沉睡，wakeup 为将别人唤醒

2.4.4 更换进程映像

1.释放子进程原来的程序和数据所占用的内存空间

2.从磁盘上找出子进程所要执行的程序和数据（通常以可执行文件的形式存放————ELF Executable and Linkable Format）

3.分配内存空间，装入新的程序和数据

4.为子进程建立初始运行环境：对各个寄存器初始化，使其返回用户态，进行该进程的程序

不同的操作系统有不同的实现方式

3 Linux 进程管理

3.1 linux 进程状态

1.运行态（TASK_RUNNING）：运行+就绪态。当前进程由运行指针所指向。

2.可中断等待态（TASK_INTERRUPTIBLE）：“浅度”睡眠，能被信号，中断或所等待资源被满足时唤醒。

3.不可中断等待态（TASK_UNINTERRUPTIBLE）：“深度”睡眠，只能在所等待资源被满足时唤醒。

4.停止态（TASK_STOPPED）：通常由于接收一个信号致使进程停止，正在被调试的进程可能处于停止态。

5.僵死态（TASK_ZOMBIE）：由于某些原因，程序被终止了，但该进程的控制结构task_struct仍然保留着。

3.2 linux 进程模式

用户模式（用户态）和内核模式（核心态）

3.2.1 用户态：当前运行的是用户程序，应用程序，或者内核之外的系统程序

3.2.2 核心态：用户态时出现系统调用或者发生中断事件（运行操作系统（核心）程序）

可以执行机器的特权指令，不受用户的干预（即使是root用户）。

根据进程的功能和运行的程序，可以将进程分为两大类

3.2.3 系统进程

只在核心态运行，执行操作系统的代码，完成管理性的工作，如内存分配和进程切换。

3.2.4 用户进程

既可以在用户态下运行，也可以在核心态下运行。

当用户进程需要进行一些需要特权的操作时（例如访问硬件、执行特权指令等），它会通过系统调用进入内核态，请求操作系统执行相关的特权操作。操作系统会在内核态中执行相应的任务，然后将控制返回给用户态。

3.3 linux 进程的结构

每个进程都有一个名为task_struct的数据结构，相当于PCB。

系统中有一个名为task的向量数组，长度默认为521B。

创建新进程时，Linux从系统内存中分配一个task_struct结构，并将它的首地址放入task中，当前运行进程的task_struct由current指针指示。

3.3.1 task_struct

与之前PCB相比额外的信息

时间和计时器：内核要记录进程的创建时间和进程运行所用的CPU时间。linux系统支持进程的时间间隔计时器。

文件系统：进程在运行时可以打开和关闭文件。task_struct结构中包含指向每个打开文件的文件描述符的指针，并且由两个指向虚拟文件系统（virtual file system VFS）索引节点的指针。第一个索引节点是进程的根目录。第二个索引节点是当前的工作目录。

两个索引节点都有一个计数字段，该计数字段记录访问该索引节点的进程数。

虚拟内存： linux系统必须了解如何将虚拟内存映射到系统的物理内存

3.3.2 系统堆栈

保存中断现场信息和进程进入核心态后执行子程序（函数）嵌套调用的返回现场信息。（在没中断的时候，它存好寄存器和函数调用的情况，中断发生后它往栈多Push一条执行到第几行的信息，在之后继续执行）

因为系统堆栈和task_struct结构存在紧密联系，两者的物理储存空间也连在一起。

内核在为每个进程分配task_struct结构的内存空间时，实际上一次分配两个连续的内存页面（8KB），底部约1KB的空间用于存放task_struct结构，上面约7KB的空间存放进程的系统堆栈。

另外，系统空间堆栈的大小是静态确定的，而用户空间堆栈可以在运行时动态扩展。

4 操作命令

4.1 有关进程

4.1.1 ps 查看进程状态

PID：进程标识号

TTY：该进程建立时所对应的终端，“？”表示该进程不占用终端

TIME：进程累计使用CPU的时间，有些运转很长时间的命令使用CPU的时间很少，所以该字段往往是00：00：00

CMD：执行进程的命令名

-a 显示系统中与TTY相关的（会话组长除外，因为不是与会话有关的）所有进程的信息

-e 显示所有进程的信息

-f 显示进程的所有信息

UID：进程属主的用户ID号

PPID：父进程的ID号

C：进程最近使用CPU的估算

STIME：进程开始时间，格式为小时：分

-l 以长格式显示进程信息

-r 只显示正在运行的进程

-u 显示面向用户的格式（包括用户名，CPU及内存使用情况，进程运行态）

-x 显示所有终端上的进程信息

4.2 kill

信号机制（也被称为软中断）是在软件层次上对中断机制的一种模拟。异步进程可以通过彼此发送信号来实现简单通信。

系统预先规定若干个不同类型的信号（x86中Linux内核设置了32种信号，现在的linux和posix.4定义了64种信号）

当进程遇到特定事件或出现特定要求时，就把一个信号写到相应进程task_struct的signal位。

接收信号的进程在运行过程中要检测自身是否收到了信号，如果已收到信号，则转去执行预先规定好的信号处理程序。处理之后，再返回原先正在执行的进程。

可以通过ctrl+c来终止进程，也可以用kill.但对于一个后台进程只能用kill来终止。

-s：指定要发送的信号，可以是信号名也可以是对应信号的编号

-p：显示进程所属的进程组号

-l：显示信号名列表，这也可以在/usr/include/linux/signal.h文件中找到

1.使用kill时如果没带信号会默认使用编号15信号杀死程序

2.撤销时kill必须是有足够的权限，撤销一个不存在的或者无权限的进程将会报错

3.终端会显示信息，有时要按enter后才显示

4.可以输入多个pid

5.可以使用kill 0 来撤销当前shell运行的所有进程，省去搜索进程号的麻烦

4.2.1 sleep

sleep 100 单位为秒

4.2.2 fork

pid_t为有符号整型量在父进程中返回子进程的PID（¿0）,在子进程中返回0，失败则返回-1

4.2.3 exec

execve才是真正执行的函数，其他都是包装过的库函数。作用为更换进程映像，根据指定文件名找到可执行文件，并用它来取代调用进程的内容。

argv和envp分别是传给被执行文件的命令行参数数组和环境变量数组

arg：命令行单个参数

path：表示完整路径

file：自动在环境变量寻找该文件

4.2.4 wait

4.2.5 exit

status：进程退出时的状态
_exit函数比exit简单，仅仅是终止进程
exit需要检查文件的打开情况，清理I/O缓存才退出

4.2.6 getpid

返回该进程pid

4.2.7 getppid

返回父进程pid

4.2.8 nice

5 系统调用的使用方式

在linux/unix系统中，系统调用和库函数都是以C函数的形式提供给用户的，头文件一般放在/usr/include/sys或者/usr/include/linux目录下。

虽然系统调用类似库函数调用，但库函数属于用户层，只能在用户态运行不能进入核心态。系统调用属于操作系统的核心层，在核心态运行，而且可以实现处理机从用户态到核心态的转变。

5.1

导航

在这里，你可以添加一些导航链接，如链接到、子节、第二节等。