LINUX源码分析——进程管理

本部分参考书籍为《深入理解LINUX内核（第三版）》，以下的内容大部分在参考书中也介绍到了，本人跟随参考书对源代码进行阅读、整理，与源代码相结合写出下列分析。LINUX版本与参考书保持一致，为2.6.11。

**1、进程的定义**

进程是程序执行时的一个实例。从内核观点看，进程的目的就是担当分配系统资源的实体。

**2、Linux系统对进程的表示**

在linux-2.6.11\include\linux\sched.h中，定义了task\_struct结构，用来描述进程。下面在task\_struct中挑几个重点的字段进行描述。

**volatile long state** 字段描述进程的状态，-1表示不能运行，0表示可以运行，大于0则表示状态停止中。在sched.h里，又定义了以下几个常量：

|  |  |  |
| --- | --- | --- |
| 状态名称 | 值 | 含义 |
| TASK\_RUNNING | 0 | 进程要么在CPU上执行，要么准备执行 |
| TASK\_INTERRUPTIBLE | 1 | 进程被挂起，知道产生硬件终端 |
| TASK\_UNINTERRUPTIBLE | 2 | 进程被挂起 |
| TASK\_STOPPED | 4 | 进程执行被暂停 |
| TASK\_TRACED | 8 | 进程执行已有调试器debugger暂停 |
| EXIT\_ZOMBIE | 16 | 进程执行被终止，但父进程还没有发布wait()类系统调用来返回有关死亡进程的信息，内核不能丢弃这一状态下的数据 |
| EXIT\_DEAD | 32 | 最终状态，父进程发出wait()类调用后执行 |

其中，后两个状态还能写在**long exit\_state**字段中

**pid\_t pid**字段存放进程标识符。新创建进程的PID通常是前一个进程的PID加1，但PID值有上限，超过上限时会循环使用已闲置的小PID号。

进程描述符存放在栈中，进程描述符之间使用双向链表连接，双向链表的结构list\_head定义在在linux-2.6.11\include\linux\list.h中:

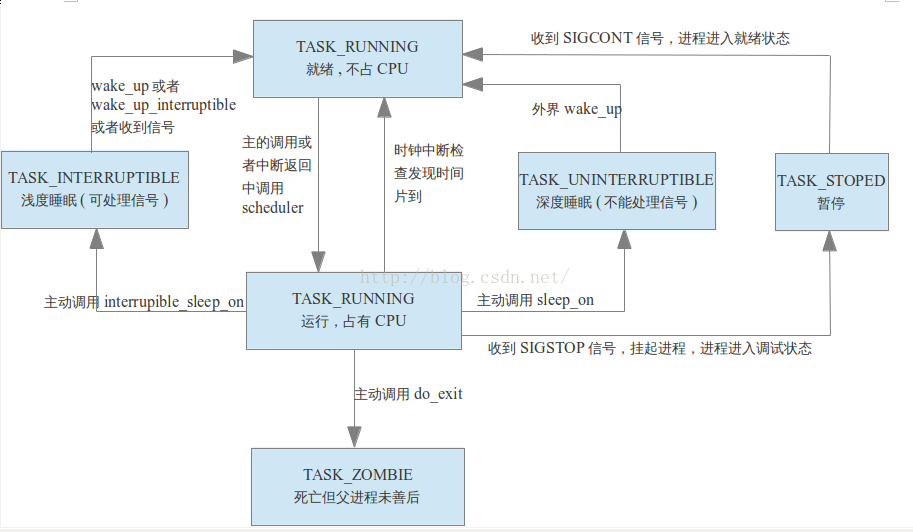
struct list\_head {

struct list\_head \*next, \*prev;

};

在task\_struct中有好几个这样的链表：**struct list\_head tasks**是所有进程间的链表；**struct list\_head run\_list**是可运行进程的链表，**struct list\_head children; struct list\_head sibling;** 用来描述进程的亲属关系等等。

**3、进程的状态转换与模式转换**

LINUX的几种状态在前面已经说明，他们之间的转换可以由下图体现：

[4]

**4、进程上下文切换机制**

（1）定义

上下文切换指内核挂起正在CPU上运行的进程，并恢复以前挂起的某个进程的执行。每个进程虽然拥有属于自己的地址空间，但所有进程都必须共享CPU寄存器；在恢复一个进程的执行之前，内核必须确保每个寄存器装入了挂起进程的值。因此（硬件）上下文就是进程恢复执行前必须装入寄存器的一组数据。

(2)源代码分析，在C:\projects\linux源码\linux-2.6.12.1\linux-2.6.11\kernel\sched.c中定义了context\_switch函数

task\_t \* context\_switch(runqueue\_t \*rq, task\_t \*prev, task\_t \*next)

{

struct mm\_struct \*mm = next->mm;

struct mm\_struct \*oldmm = prev->active\_mm;

if (unlikely(!mm)) {

next->active\_mm = oldmm;

atomic\_inc(&oldmm->mm\_count);

enter\_lazy\_tlb(oldmm, next);

} else

switch\_mm(oldmm, mm, next);

if (unlikely(!prev->mm)) {

prev->active\_mm = NULL;

WARN\_ON(rq->prev\_mm);

rq->prev\_mm = oldmm;

}

/\* Here we just switch the register state and the stack. \*/

switch\_to(prev, next, prev);

return prev;

}

这个函数中，前面的if部分用来切换到新的地址空间，后面调用switch\_to用来切换寄存器和栈中的内容，switch\_to函数是硬件相关的，不同平台不一样。

**5、软中断**

软中断指的是正在执行中的进程发出的由CPU执行某些任务的请求。在linux-2.6.11\include\linux\interupt.h中，定义了以下几种软中断

enum

{

HI\_SOFTIRQ=0, //处理高优先级的tasklet

TIMER\_SOFTIRQ, //处理和时钟中断相关的tasklet

NET\_TX\_SOFTIRQ, //把数据包发送到网卡

NET\_RX\_SOFTIRQ, //从网卡接收数据包

SCSI\_SOFTIRQ, //SCSI命令的后台中断处理

TASKLET\_SOFTIRQ //处理常规tasklet

};

我们自己定义的软中断，归于最后一种tasklet里面，优先级最低。

**extern void open\_softirq(int nr, void (\*action)(struct softirq\_action\*), void \*data);**

用来处理软中断的初始化，三个参数分别是软中断下表、指向要执行的软中断函数以及可能由软中断函数使用的数据结构的指针

**void raise\_softirq(unsigned int nr)**负责触发本地CPU上的软中断

**fastcall unsigned int \_\_do\_IRQ(unsigned int irq, struct pt\_regs \*regs)**定义在linux-2.6.11\kernel\irq\handle.c中，负责执行软中断

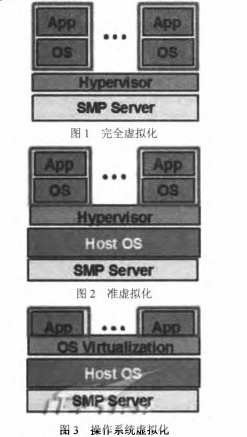
**6、进程调度**

Linux进程调度基于分时技术，CPU时间被分成一个个时间片，进程在运行时占用该时间片。调度策略根据进程的优先级对它们进行分类。具体算法又与进程是普通进程或是实时进程而不同。

浅析操作系统新技术——操作系统虚拟化

·1、定义

在计算机领域，虚拟化是将计算机的各种实体资源进行抽象与转换后呈现出来的资源管理技术。本质上，虚拟化就是由位于下层的软件模块箱上一层软件模块提供一个与它原先所期待的运行环境完全一致的接口，使得上层软件可以直接运行在下一层软件所提供的环境上，将一份资源抽象成多份，也可以将多份资源抽象成一份。[1]。虚拟化技术打破了实体结构间的障碍，使用户能够比原来的组态更好的应用系统资源;运用虚拟化技术，可以将性能过剩的高性能硬件或者性能过低的老旧硬件重组重用，更高效地运用资源。

虚拟化技术分为硬件虚拟化、存储虚拟化、网络虚拟化、引用虚拟化等；狭义上，虚拟化常常指的是硬件虚拟化或者是虚拟机，指的是由虚拟化层提供独立的计算机系统，在上层软件看来，每一个虚拟机就是一个真实的机器。硬件虚拟化技术又分为完全虚拟化、准虚拟化与操作系统虚拟化三种。三种虚拟化方式的区别，在于实现虚拟化的中间层(hypervisor)在系统中的层次的高低。如下图所示，完全虚拟化中，hypervisor直接管理调用硬件资源，虚拟机运行在hypervisor上；在准虚拟化中，hypervisor运行在一个传统的操作系统上，是主机操作系统的一个应用软件；操作系统虚拟化没有独立的hypervisor层，由主机操作系统本身负责在多个虚拟服务器之间分配硬件资源。

在前两种虚拟化中，各个虚拟机可以运行不同的操作系统，各个系统间相互独立；在操作系统虚拟化中，虽然各个虚拟机的系统间的运行也是相互独立的，但是它们都必须运行同一种操作系统。

2、操作系统虚拟化的优缺点

前文已经提到，虚拟化能够对资源进行重组重用，更高效地利用计算资源。而相对于传统的基于hypervisor的虚拟化，操作系统虚拟化的优点有：

1. 操作系统虚拟化的效率更高。在完全虚拟化与准虚拟化中，hypervisor必须对硬件进行虚拟；而操作系统虚拟化不需要对硬件进行仿真，而是共享宿主机的内核，由宿主机操作系统调用硬件资源，因此对硬件资源的利用率非常高，接近于物理机的性能；
2. 操作系统虚拟化的启动更方便。基于hypervisor的虚拟化启动时间长，操作系统层虚拟化启动时间则非常短；
3. 操作系统虚拟化更利于共享。操作系统虚拟化的每一个容器因为不含有完整的操作系统，因此非常小，一般为MB级，有利于传输、共享；而完全虚拟化或者半虚拟化里每一个虚拟机都含有完整的操作系统，占用空间容量一般为GB级，不利于共享

当然，相对于传统的虚拟化，操作系统虚拟化也有一些缺点:

1. 操作系统虚拟化的环境依赖于宿主操作系统，每一个容器运行时的操作系统必须与宿主操作系统相同；而完全虚拟化与半虚拟化的虚拟机内可以运行不同于宿主的操作系统。例如，在Win10操作系统的虚拟机上运行Linux或者WinXP操作系统等；
2. 相比于基于hypervisor的虚拟化，操作系统虚拟化实例之间的隔离程度不够，因此安全性不如完全的虚拟化。

3、操作系统虚拟化的应用场景

1.方便代码流水线管理

在软件开发过程中，开发——测试——生产环境的基础硬件都稍有不同，虚拟化技术可以创建从开发到生产一致的运行环境。如果利用传统的虚拟机技术，在不同环节传输的数据量较大、各个环节启动的速度慢、运行时耗费资源较大，不利于实际的应用；而操作系统虚拟化中容器占用的数据空间小、启动速度快、运行时耗费资源少，适合代码流水线不同环节的共享，有利于软件开发

2．方便软件开发环境的部署。

软件开发环境的部署往往耗时长、易出错，在不同机器上不易保持统一。基于操作系统虚拟化进行软件开发环境的部署更利于保持软件开发环境间的统一，减少部署开发环境耗费的时间和精力[2]

1. 方便微服务架构的使用

利用操作系统虚拟化，可以讲传统分布式服务继续拆分解耦，形成一些更小的服务模块，每个服务模块运行在一个小容器上，使整个系统更利于维护与敏捷开发。[3]

其它的应用还有将操作系统虚拟化应用在Web服务器上、应用在一次性应用上等等

4、对操作系统虚拟化的典型——Docker进行简要分析

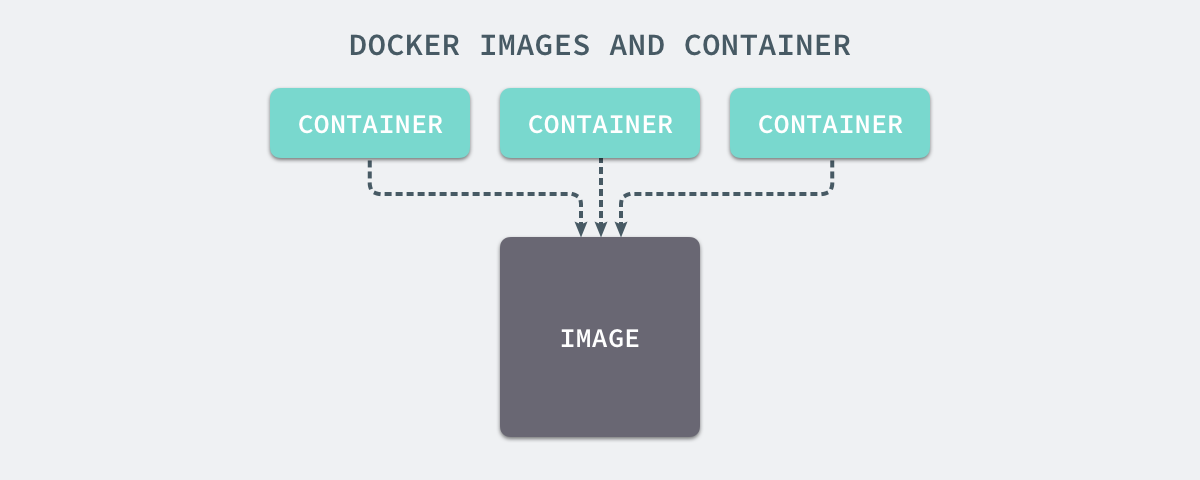
目前使用最广泛的操作系统虚拟化技术是Docker。它是基于Go语言的一款开源的操作系统虚拟化容器引擎。下面我们主要对Docker的原理进行简要的分析。

Docker的实现基于Linux提供的命名空间（Namespaces）、控制组（Control Groups）以及联合文件系统三种方法。

命名空间是Linux提供的用于分离进程树、网络接口、挂载点以及进程间通信等资源的方法。Docker通过Linux的命名空间机制对不同的容器实现了隔离。

Linux的命名空间为不同的容器隔离了文件系统、网络并与宿主机器之间的晋城相互隔离，但命名空间并不能够进行物理资源上的隔离，如果一个容器占用了过多的硬件资源，就会对整体的效率产生影响。而Control Groups就提供了对宿主机器上的物理资源上的隔离，例如CPU、内存、网络带宽等等

另外，为了实现容器的快速部署与应用，Docker采用了联合文件系统，讲Docker的文件系统分为镜像(image)与容器（container）两个部分，镜像是只读的，而容器是可以修改的，多个容器可以共享同一个镜像；从而在进行Docker的分享时，可以将镜像打包复用、快速部署。这一机制就是联合文件系统。



[1]修长虹,梁建坤,辛艳.虚拟化技术综述[J].网络安全技术与应用,2016,(5):18-19. DOI:10.3969/j.issn.1009-6833.2016.05.012.

[2] 王飞. 基于Docker的研发部署管理平台的设计与实现[D]. 北京交通大学, 2015.

[3] 《浅析Docker容器的应用场景》<http://www.dockone.io/article/1282>

[4] https://blog.csdn.net/kklvsports/article/details/52268085