闫明 极客来 GeekCome

欢迎直接访问个人博客 http://blog.geekcome.com

闫明

虚拟化–操作系统级 LXC Linux Containers内核轻量级虚拟化技术

分类： 技术手册 Linux内核游记 云计算 2014-05-16 22:31 1175人阅读 评论(0) 收藏 举报

虚拟化技术 虚拟化 内核 linux lxc

目录(?) [+]

友情提示：非原文链接可能会影响您的阅读体验，欢迎查看原文。(http://blog.geekcome.com)
原文地址：http://blog.geekcome.com/archives/288

软件平台：Ubuntu 14.04

容器有效地将由单个操作系统管理的资源划分到孤立的组中，以更好的在孤立的组之间有冲突的资

源使用需求。与其他的虚拟化比较，这样既不需要指令级模拟，也不需要即时编译。容器可以在寒心CPU本地运行指令，而不需要任何专门的解释机制。此外半虚拟化和系统调用替换的复杂性。

LXC的实现是基于内核中的namespace和cgroup实现的。

namespace：

和C++中的namespace概念相似。在Linux操作系统中，系统资源如：进程、用户账户、文件系统、网络都是属于某个namespace。每个namespace下的资源对于其他的namespace资源是透明的，不可见的。因为在操作系统层上就会出现相同的pid的进程，多个相同uid的不同账号。

内核中的实现：

namespace是针对每一个进程而言的，所以在task_struct结构的定义中有一个指向nsproxy的指针

```
1 /* namespaces */
2 struct nsproxy *nsproxy;
```

该结构体的定义如下：

```
01 /*
02  * A structure to contain pointers to all per-process
03  * namespaces - fs (mount), uts, network, sysvipc, etc.
04  *
05  * The pid namespace is an exception -- it's accessed using
06  * task_active_pid_ns.  The pid namespace here is the
07  * namespace that children will use.
08  *
09  * 'count' is the number of tasks holding a reference.
10  * The count for each namespace, then, will be the number
11  * of nsproxies pointing to it, not the number of tasks.
12  *
13  * The nsproxy is shared by tasks which share all namespaces.
14  * As soon as a single namespace is cloned or unshared, the
15  * nsproxy is copied.
```

Linux内核学习笔记

文章：53篇

阅读：177505

ARM-Linux驱动移植

文章：19篇

阅读：73986

ARM-Linux驱动开发

文章：17篇

阅读：84684

阅读排行


```
 */
struct nsproxy {
    atomic_t count;
    struct uts_namespace *uts_ns;
    struct ipc_namespace *ipc_ns;
    struct mnt_namespace *mnt_ns;
    struct pid_namespace *pid_ns_for_children;
    struct net       *net_ns;
};
```

其中第一个属性count表示的是该命名空间被进程引用的次数。后面的几个分别是不同类型的命名空间。以pid_namespace为例。

其结构如下所示：

```
struct pid_namespace {
    struct kref kref;//引用计数
    struct pidmap pidmap[PIDMAP_ENTRIES];//用于标记空闲的id号
    struct rcu_head rcu;
    int last_pid;//上一次分配的id号
    unsigned int nr_hashed;
    struct task_struct *child_reaper;//相当于全局的init进程，用于对僵尸进程进行
    struct kmem_cache *pid_cachep;
    unsigned int level;//namespace的层级
    struct pid_namespace *parent;//上一级namespace指针
#ifdef CONFIG_PROC_FS
    struct vfsmount *proc_mnt;
    struct dentry *proc_self;
#endif
#ifdef CONFIG_BSD_PROCESS_ACCT
    struct bsd_acct_struct *bacct;
#endif
    struct user_namespace *user_ns;
    struct work_struct proc_work;
    kgid_t pid_gid;
    int hide_pid;
    int reboot; /* group exit code if this pidns was rebooted */
    unsigned int proc_inum;
```

```
24 };
```

内核中的pid结构表示：

```
1 struct pid
2 {
3     atomic_t count;
4     unsigned int level;//pid对应的级数
5     /* lists of tasks that use this pid */
6     struct hlist_head tasks[PIDTYPE_MAX];//一个pid可能对应多个task_struct
7     struct rcu_head rcu;
8     struct upid numbers[1];//该结构是namespace中的具体的pid，从1到level各级别的
  只不过不需要再分配空间
9 };
```

上面的结构体就是内核中进程的标示符，可以用于标识内核中的tasks、process groups和sessions。这个结构体和具体的task通过hash来关联，通过具体的task对应的pid的值可以获得绑定的pid结构体。

属于具体的namespace的pid结构upid：

```
1 struct upid {
2     /* Try to keep pid_chain in the same cacheline as nr for find_vpid */
3     int nr;
4     struct pid_namespace *ns;
5     struct hlist_node pid_chain;
6 };
```

该结构体是用来获得结构体pid的具体的id，它只对特定的namespace可见。会通过函数find_pid_ns(int nr,pid_namespace *ns)函数来获得具体的PID结构。

整体结构如下图：

2014年09月 (1)
2014年05月 (4)
2014年04月 (7)
2014年03月 (2)
2014年02月 (1)
展开

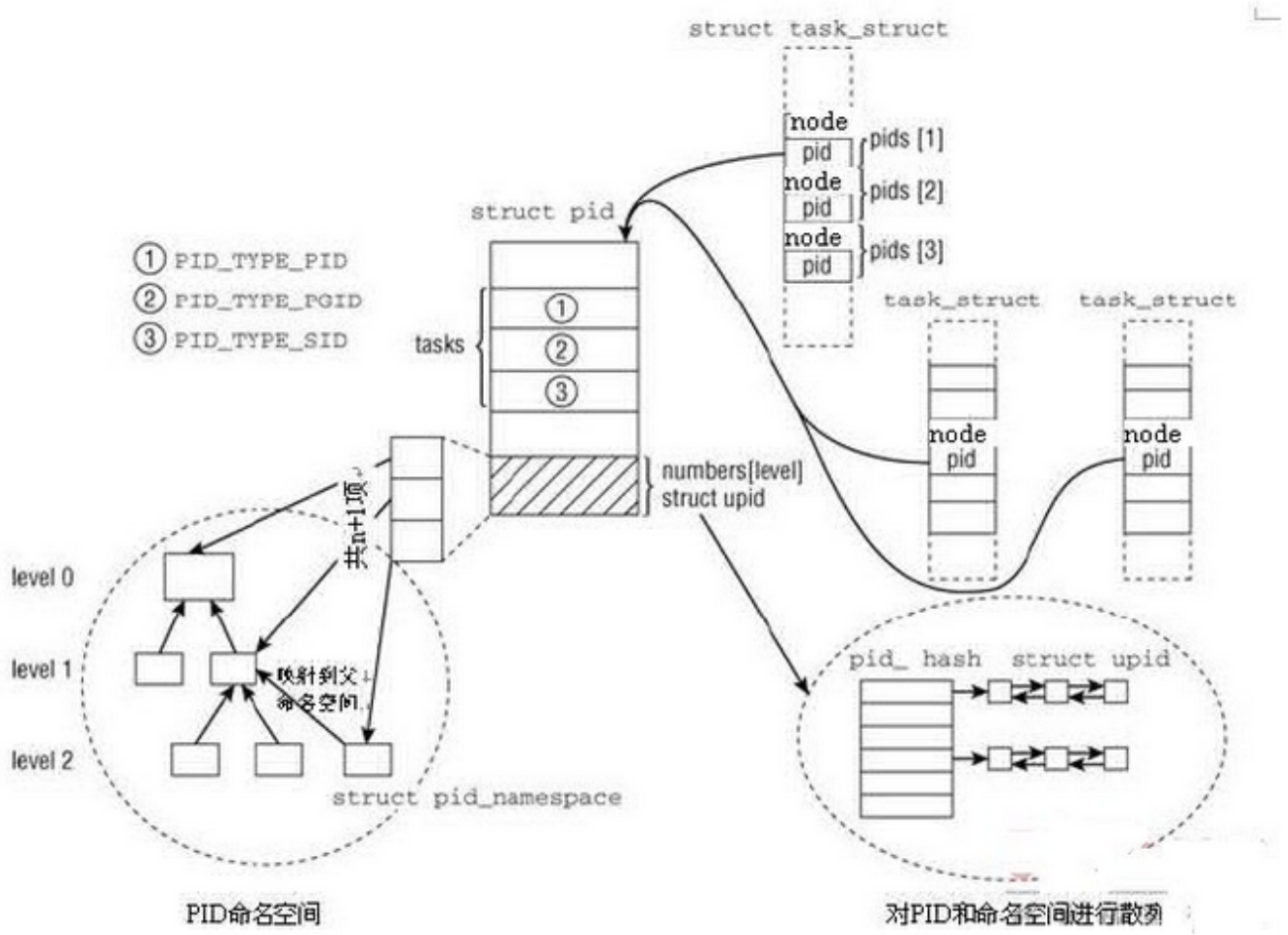
评论排行

Google、 百度、谷歌的 (51)
C语言中的二级指针(双指 (47)
Linux内核--基于Netfilter (33)
void及void指针含义的深 (25)
linux内核移植-移植2.6.3 (22)
推荐系统--基于用户的协 (14)
Linux内核--网络栈实现分 (14)
Windows 就不能改善一下 (13)
程序员都能看懂的内涵小 (12)
Linux内核--网络协议栈深 (11)

最新评论

Linux0.11内核--启动引导代码分
ioscoder: 我调试发现，ljmp $INITSEG, $go 这句好像没有跳转成功，后面的代码都没执行希望楼主帮...

Linux0.11内核--系统中断处理程
HQ_Yuan: 大神，你好！ 能请教下吗？ 用户态通过系统调用切换到内核态，是将特权等级从r3提升到r0，在sys...



Cgroup：

Cgroup是control groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程组所使用的物理资源（CPU，内存，IO等等）的机制。Cgroup也是LXC位实现虚拟化所使用的资源管理的手段。可以说

没有Cgroup就没有LXC，也就没有Docker。

Cgroup提供的功能：

- 限制进程组可以使用的资源数量。一单进程组使用的内存达到限额就会引发异常
- 控制进程组的优先级。可以使用cpu子系统为某个进程组分配特定的cpu share
- 记录进程组使用资源的数量
- 进程组隔离。eg.使用ns子系统可以使不同的进程组使用不同的namespace，已达到
- 进程组控制

Cgroup子系统：

- blkio：设定输入输出限制
- cpu：使用调度程序对CPU的Cgroup任务访问
- cpuacct：自动生成Cgroup任务所使用的CPU报告
- cpuset：为Cgroup中的任务分配独立的CPU（多核系统中）和内存节点
- devices：允许或拒绝Cgroup中的任务访问设备
- freezer：挂起或回复Cgroup中的任务
- memory：Cgroup中任务使用内存的限制
- net_cls：允许Linux流量控制程序识别从cgroup中生成的数据包
- ns：命名空间子系统

Cgroup中的概念：

- 任务（Task）：任务就是系统中的一个进程
- 控制族群（control group）：一组按某种标准划分的进程，控制族群通常按照应用划分，即与某应用相关的一组进程，被划分位一个进程组（控制族群）。Cgroup中资源控制都是以控制族群为单位实现。一个进程可以加入某个控制族群，也可以从一个进程组迁移到另一个控制族群。
- 层级：控制族群可以组织成层级的形式–控制族群树。
- 子系统：资源控制器，比如CPU子系统就是控制CPU时间分配的一个控制器。子系统必须attach到

一个层级上才能起作用，一个子系统附件到某个层级以后，这个层级上的所有控制族群都收到这个子系统的控制。

Cgroup使用控制CPU：

在Ubuntu中，cgroup默认挂载位置/sys/fs/cgroup目录。ls查看一下：

```
yan@yan-Z400:/sys/fs/cgroup$ ls
blkio      cpu      cpuset   freezer  memory      systemd
cgmanager  cpuacct  devices  hugetlb  perf_event
```

可以看到cgroup的不同子系统目录。

在CPU文件夹中新建一个geekcome目录，默认ubuntu已经将子系统全部挂载了：

进入cpu文件夹新建一个geekcome文件夹，然后查看：

```
yan@yan-Z400:/sys/fs/cgroup/cpu$ ls
cgroup.clone_children  cgroup.sane_behavior  cpu.shares  lxc
cgroup.event_control   cpu.cfs_period_us     cpu.stat    notify_on_releas
cgroup.procs           cpu.cfs_quota_us      geekcome    release_agent
```

新建文件夹后在文件夹里会自动生成相应的文件：

```
geekcome
├── cgroup.clone_children
├── cgroup.event_control
├── cgroup.procs
├── cpu.cfs_period_us
├── cpu.cfs_quota_us
├── cpu.shares
├── cpu.stat
├── notify_on_release
└── tasks
```

下面就跑一个死循环程序，导致CPU使用率到达100%。

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
5046 yan       20   0   25928   4848   2324 R 100.0  0.1   0:22.47 pythor
```

现在执行如下的命令：

```
echo "50000" >/sys/fs/cgroup/cpu/geekcome/cpu.cfs_quota_us
echo "5046" >/sys/fs/group/cpu/geekcome/tasks
```

再top查看一下：

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
5046 yan       20   0   25928   4844   2324 R  49.8  0.1   0:49.27 pythor
```

进程5046的cpu使用率从100%降低到50%。在Cgroup里，可以使用cpu.cfs_period_us和cpu.cfs.quota_ua来限制在单位时间里可以使用的cpu时间。这里cfs的含义是Completely Fair Scheduler（完全公平调度器）。cpu.cfs_period_us是时间周期，默认是100000（百毫秒）。cpu.cfs_quota_us是在这期间可以使用的cpu时间，默认-1（无限制）。

在上面的实例中，通过修改cpu.cfs_period_us文件，将百毫秒修改为一半，成功将CPU使用率降低到50%。cfs.quota_us文件主要对于多核的机器，当有n个核心时，一个控制组的进程最多能用到n倍的cpu时间。

Cgroup除了资源控制功能外，还有资源统计功能。云计算的按需计费可以通过它来实现。这里只实例CPU的控制，其他的子系统控制请自行实验。

LXC使用：

创建一个容器：

```
lxc-create -n name [-f config_file] [-t template]
sudo lxc-create -n ubuntu01 -t ubuntu
```

-n就是虚拟机的名字，-t是创建的模板，保存路径在/usr/lib/lxc/templates。模板就是一个脚本文件，

执行一系列安装命令和配置（穿件容器的挂载文件系统，配置网络，安装必要软件，创建用户并设置密码等）。

显示已经创建的容器：

```
1 lxc-ls
```

启动一个容器：

```
1 lxc-start -n name [-f config_file] [-s KEY=VAL] [command]
```

启动一个容器，可以指定要执行的命令，如果没有指定，lxc-start会默认执行/sbin/init命令，启动这个容器。

关闭一个容器：

```
1 lxc-stop -n name
```

快速启动一个任务，任务执行完毕后删除容器：

```
1 lxc-execute -n name [-f config_file] [-s KEY=VAL ] [--] command
```

它会按照配置文件执行lxc-create创建容器，如果没有指定的配置文件，则选择默认。该命令一般用于快速使用容器环境执行摸个任务，任务执行完毕后删除掉容器。

(完)

给出原文连接。

如果对文章有任何问题，都可以在评论中留言，我会尽可能的答复您，谢谢你的阅读

上一篇 分布式数据库中间件–(3) Cobar对简单select命令的处理过程

下一篇 (译)可视化垃圾收集(GC)算法

主题推荐 虚拟化技术 操作系统 虚拟化 内核 linux

猜你在找

全面解析虚拟化技术——从网格到操作系统的多线程

Docker: 现在和未来

linux虚拟化技术xen的架设，配置

Linux系统虚拟化技术详解，部署及调优（不含认证）-

Linux&VMware&vBox&虚拟化技术资料汇总

开源虚拟化操作系统OSv初探

RH318之安装虚拟化操作系统

操作系统虚拟化底层基础之命名空间（namespace）

操作系统对硬件虚拟化的影响

基于容器的虚拟化lxc

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap