深入理解 Linux的进程，线程，PID，LWP，TID，TGID

在Linux的top和ps命令中，默认看到最多的是pid (process ID)，也许你也能看到lwp (thread ID)和tgid (thread group ID for the thread group leader)等等，而在Linux库函数和系统调用里也许你注意到了pthread id和tid等等。还有更多的ID，比如pgrp (process group ID), sid (session ID for the session leader)和 tpgid (tty process group ID for the process group leader)。概念太多可能很晕，但是只要对Linux的进程和线程的基本概念有准确的理解，这些ID的含义都迎刃而解。下面将介绍进程和线程的核心概念，并以一个示例程序来验证这些ID之间的关系。

Linux的进程和线程

Linux的进程和线程有很多异同点，可以Google下。但只要能清楚地理解一下几点，则足够理解Linux中各种ID的含义。

- **进程是资源分配的基本单位，线程是调度的基本单位**
- 进程是资源的集合，这些资源包括内存地址空间，文件描述符等等，一个进程中的多个线程共享这些资源。
- CPU对任务进行调度时，可调度的基本单位 (dispatchable entity)是线程。如果一个进程中没有其他线程，可以理解成这个进程中只有一个主线程，这个主进程独享进程中的所有资源。
- 进程的个体间是完全独立的，而线程间是彼此依存，并且共享资源。多进程环境中，任何一个进程的终止，不会影响到其他非子进程。而多线程环境中，父线程终止，全部子线程被迫终止(没有了资源)。

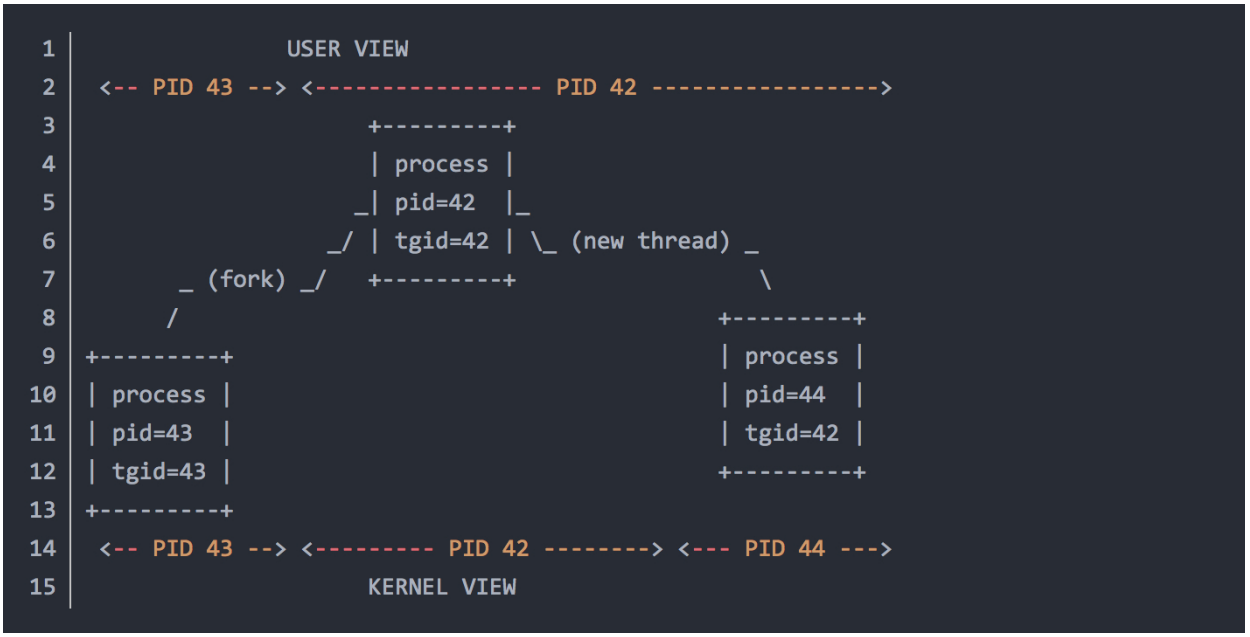
上述第一点说明是最基础的，也是最重要的。

初步理解各种ID。基本上按照重要程度从高到低，在分割线下方的IDs不太重要。

- **pid**: 进程ID。
- **lwp**: 线程ID。在用户态的命令(比如ps)中常用的显示方式。
- **tid**: 线程ID，等于lwp。tid在系统提供的接口函数中更常用，比如syscall(SYS_gettid)和syscall(__NR_gettid)。
- **tgid**: 线程组ID，也就是线程组leader的进程ID，等于pid。
- ------分割线------
- **pgid**: 进程组ID，也就是进程组leader的进程ID。
- **pthread id**: pthread库提供的ID，生效范围不在系统级别，可以忽略。
- **sid**: session ID for the session leader。
- **tpgid**: tty process group ID for the process group leader。

从上面的列表看出，各种ID最后都归结到pid和lwp(tid)上。所以理解各种ID，最终归结为理解pid和lwp(tid)的联系和区别。

下面的图是一张描述父子进程，线程之间关系的图。



上图很好地描述了用户视角(user view)和内核视角(kernel view)看到线程的差别：

- 从用户视角出发，在pid 42中产生的tid 44线程，属于tgid(线程组leader的进程ID) 42。甚至用ps和top的默认参数，你都无法看到tid 44线程。
- 从内核视角出发，tid 42和tid 44是独立的调度单元，可以把他们视为"pid 42"和"pid 44"。

需要指出的是，有时候在Linux中进程和线程的区分也是不是十分严格的。即使线程和进程混用，pid和tid混用，根据上下文，还是可以清楚地区分对方想要表达的意思。上图中，从内核视角出发看到了pid 44，是从调度单元的角度出发，但是在top或ps命令中，你是绝对找不到一个pid为44的进程的，只能看到一个lwp(tid)为44的线程。

理解pid和lwp(tid)的示例程序

下面利用一个示例程序来进一步理解pid和lwp(tid)，以及利用格式化的ps命令打印出各种ID。下面的程序在main函数中创建了2个子线程，加上main函数这个主线程，一共有3个线程。在3个线程中分别打印pthread id, pid和lwp(tid)，来验证pid和lwp(tid)的关系。

```
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <pthread.h>
#define gettidv1() syscall(__NR_gettid) // new form
#define gettidv2() syscall(SYS_gettid)  // traditional form
void *ThreadFunc1()
{
        printf("the pthread_1 id is %ld\n", pthread_self());
        printf("the thread_1's Pid is %d\n", getpid());
        printf("The LWPID/tid of thread_1 is: %ld\n", (long int)gettidv1());
        pause();
        return 0;
```

```
}
void *ThreadFunc2()
{
        printf("the pthread_2 id is %ld\n", pthread_self());
        printf("the thread_2's Pid is %d\n", getpid());
        printf("The LWPID/tid of thread_2 is: %ld\n", (long int)gettidv1());
        pause();
        return 0;
}
int main(int argc, char *argv[])
{
        pid_t tid;
        pthread_t pthread_id;
        printf("the master thread's pthread id is %ld\n", pthread_self());
        printf("the master thread's Pid is %d\n", getpid());
        printf("The LWPID of master thread is: %ld\n", (long int)gettidv1());
        // 创建2个线程
        pthread_create(&pthread_id, NULL, ThreadFunc2, NULL);
        pthread_create(&pthread_id, NULL, ThreadFunc1, NULL);
        pause();
        return 0;
}
```

注意编译的时候要利用-l指定library参数。

```
# gcc threadTest.c -o threadTest -l pthread
```

执行程序，结果如下：

```
# ./threadTest
the master thread's pthread id is 140154481125184
the master thread's Pid is 20992
The LWPID of master thread is: 20992
the pthread_1 id is 140154464352000
the thread_1's Pid is 20992
The LWPID/tid of thread_1 is: 20994
the pthread_2 id is 140154472744704
the thread_2's Pid is 20992
The LWPID/tid of thread_2 is: 20993
```

上述结果说明pthread id是pthread库提供的ID，在系统级别没有意义。pid都是线程组leader的进程ID，即20992。而lwp(tid)则是线程ID，分别是20993和20994。

同时利用ps来查看结果，注意ps默认只打印进程级别信息，需要用-L选项来查看线程基本信息。

```
# ps -eo pid,tid,lwp,tgid,pgrp,sid,tpgid,args -L | awk '{if(NR==1) print $0; if($8~/threadTest/) print $0}'
  PID   TID   LWP  TGID  PGRP   SID TPGID COMMAND
20992 20992 20992 20992 20992 30481 20992 ./threadTest
20992 20993 20993 20992 20992 30481 20992 ./threadTest
20992 20994 20994 20992 20992 30481 20992 ./threadTest
```

从上述结果中可以看到：

- PID=TGID: 20992
- TID=LWP: 20993 or 20994
- 至于SID，30481是bash shell的进程ID。

Linux用户态命令查看线程

top

默认top显示的是task数量，即进程。

```
top - 16:58:04 up 44 days,  5:14,  1 user,  load average: 1.09, 1.08, 1.05
Tasks:  79 total,   2 running,  77 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.2 us, 43.9 sy,  0.0 ni, 49.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem :  8010456 total,  7265972 free,   133236 used,   611248 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  7622656 avail Mem
```

可以利用敲"H"，来切换成线程。如下，可以看到实际上有96个线程。也可以直接利用top -H命令来直接打印线程情况。

```
top - 16:57:15 up 44 days,  5:13,  1 user,  load average: 1.05, 1.07, 1.05
Threads:  96 total,   3 running,  93 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.7 us, 44.2 sy,  0.0 ni, 49.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem :  8010456 total,  7266512 free,   132924 used,   611020 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  7623132 avail Mem
```

ps

ps的-L选项可以看到线程，通常能打印出LWP和NLWP相关信息。如下命令即可查看线程信息：

```
ps -eLf
```

pidstat

pidstat -t [-p pid号] 可以打印出线程之间的关系。

htop

要在htop中启用线程查看，开启htop，然后按<F2>来进入htop的设置菜单。选择“设置”栏下面的“显示选项”，然后开启“树状视图”和“显示自定义线程名”选项。按<F10>退出设置。

注：MAC的F2按fn+F2。

LINUX中查看进程的多线程

06/07/2012MASTER 4 COMMENTS

在SMP系统中，我们的应用程序经常使用多线程的技术，那么在Linux中如何查看某个进程的多个线程呢?

本文介绍3种命令来查看Linux系统中的线程(LWP)的情况:

在我的系统中，用qemu-system-x86_64命令启动了一个SMP的Guest，所以有几个qemu的线程，以此为例来说明。

1. pstree 命令，查看进程和线程的树形结构关系。

```
[root@jay-linux ~]# pstree | grep qemu
     |-gnome-terminal-+-bash---qemu-system-x86---2*[{qemu-system-x8}]
[root@jay-linux ~]# pstree -p | grep qemu
        |-gnome-terminal(10194)-+-bash(10196)---qemu-system-x86(10657)-+-{qemu-sys
        |                       |                                      `-{qemu-sys
```

2. ps 命令，-L参数显示进程，并尽量显示其LWP(线程ID)和NLWP(线程的个数)。

```
[root@jay-linux ~]# ps -eLf | grep qemu
root     10657 10196 10657  0    3 13:48 pts/1    00:00:00 qemu-system-x86_64 -hda
root     10657 10196 10660  3    3 13:48 pts/1    00:00:26 qemu-system-x86_64 -hda
root     10657 10196 10661  2    3 13:48 pts/1    00:00:19 qemu-system-x86_64 -hda
root     10789  9799 10789  0    1 14:02 pts/0    00:00:00 grep --color=auto qemu
```

上面命令查询结果的第二列为PID，第三列为PPID，第四列为LWP，第六列为NLWP。

另外，**ps命令还可以查看线程在哪个CPU上运行**，命令如下:

```
[root@jay-linux ~]# ps -eo ruser,pid,ppid,lwp,psr,args -L | grep qemu
root     10657 10196 10657   1 qemu-system-x86_64 -hda smep-temp.qcow -m 1024 -smp
root     10657 10196 10660   1 qemu-system-x86_64 -hda smep-temp.qcow -m 1024 -smp
root     10657 10196 10661   2 qemu-system-x86_64 -hda smep-temp.qcow -m 1024 -smp
root     10834  9799 10834   1 grep --color=auto qemu
```

其中，每一列依次为：用户ID，进程ID，父进程ID，线程ID，运行该线程的CPU的序号，命令行参数（包括命令本身）。

3. top 命令，其中H命令可以显示各个线程的情况。（在top命令后，按H键；或者top -H）

```
[root@jay-linux ~]# top -H
top - 14:18:20 up 22:32,  4 users,  load average: 2.00, 1.99, 1.90
Tasks: 286 total,   1 running, 285 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3943892k total,  1541540k used,  2402352k free,   164404k buffers
Swap:  4194300k total,        0k used,  4194300k free,   787768k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10660 root      20   0 1313m 188m 2752 S  2.3  4.9   0:46.78 qemu-system-x86
10661 root      20   0 1313m 188m 2752 S  2.0  4.9   0:39.44 qemu-system-x86
10867 root      20   0 15260 1312  960 R  0.3  0.0   0:00.07 top
    1 root      20   0 19444 1560 1252 S  0.0  0.0   0:00.34 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.02 kthreadd
....
```

在top中也可以查看进程（进程）在哪个CPU上执行的。

执行top后，按f，按j（选中* J: P = Last used cpu (SMP)），然后按空格或回车退出设置，在top的显示中会多出P这一列是最近一次运行该线程（进程）的CPU.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  P COMMAND
10661 root      20   0 1313m 188m 2752 S  2.3  4.9   0:44.24 3 qemu-system-x86
10660 root      20   0 1313m 188m 2752 S  2.0  4.9   0:51.74 0 qemu-system-x86
10874 root      20   0 15260 1284  860 R  0.7  0.0   0:00.32 2 top
    1 root      20   0 19444 1560 1252 S  0.0  0.0   0:00.34 0 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.02 1 kthreadd
```

更多信息，请 man pstree, man top, man ps 查看帮助文档。

注: LWP为轻量级进程（即：线程），(light weight process, or thread) 。