Linux Namespaces机制提供一种资源隔离方案。当调用clone时，设定了CLONE\_NEWPID，就会创建一个新的PID Namespace，clone出来的新进程将成为Namespace里的第一个进程。一个PID Namespace为进程提供了一个独立的PID环境，PID Namespace内的PID将从1开始，在Namespace内调用fork，vfork或clone都将产生一个在该Namespace内独立的PID。新创建的Namespace里的第一个进程在该Namespace内的PID将为1，就像一个独立的系统里的init进程一样。该Namespace内的孤儿进程都将以该进程为父进程，当该进程被结束时，该Namespace内所有的进程都会被结束。PID Namespace是层次性，新创建的Namespace将会是创建该Namespace的进程属于的Namespace的子Namespace。子Namespace中的进程对于父Namespace是可见的，一个进程将拥有不止一个PID，而是在所在的Namespace以及所有直系祖先Namespace中都将有一个PID。系统启动时，内核将创建一个默认的PID Namespace，该Namespace是所有以后创建的Namespace的祖先，因此系统所有的进程在该Namespace都是可见的。

在新Namespace里的进程不能向其他域的进程发送信号，高层次的Namespace可以向子孙Namespace发送信号。

#ifndef CLONE\_NEWPID

#define CLONE\_NEWPID 0x20000000

#endif

int do\_newpidns(void)

{

register pid\_t pid, waited;

int status;

pid = syscall(SYS\_clone, CLONE\_NEWPID | SIGCHLD, 0, 0, 0);

switch (pid)

{

case -1:

perror("clone");

return -1;

/\* child: we are pid number 1 in the new namespace \*/

case 0:

/\* We add an extra check for old kernels because sys\_clone() doesn't

EINVAL on unknown flags \*/

if (getpid() == 1)

{

return 0;

}

else

return -1;

default:

/\* Let's wait for our child \*/

waited = waitpid(pid, &status, 0);

if (waited != pid)

{

perror("waitpid");

exit(EXIT\_FAILURE);

}

else

{

printf("child exit ok!\n");

/\* FIXME: we proxy the exit code, but only if the child terminated normally \*/

if (WIFEXITED(status))

exit(WEXITSTATUS(status));

exit(EXIT\_SUCCESS);

}

}

}

int main()

{

int ret；

ret = do\_newpidns();

assert(ret == 0);

printf("the child pid is %d\n",getpid());

printf("the father's pid is %d\n",getppid());

printf("uid:%d, euid:%d, gid:%d, egid:%d\n", getuid(), geteuid(), getgid(), getegid());

return 0;

}