# 第 0006 讲 4 进程管理 4 大常用 API 案例分析

一、find_get_pid(...)函数功能：根据进程编号获取对应的进程描述符，具体 Linux 内核源码对应函数设计如下：

```
kernel > C pid.c > find_get_pid(pid_t)
446 struct pid *find_get_pid(pid_t nr)
447 {
448     struct pid *pid;
449
450     rcu_read_lock();
451     pid = get_pid(find_vpid(nr));
452     rcu_read_unlock();
453
454     return pid;
455 }
456 EXPORT_SYMBOL_GPL(find_get_pid);
457
```

二、pid_task(...)函数功能：获取任务的任务描述符数据信息，具体 Linux 内核源码对应函数设计如下：

```
kernel > C pid.c > pid_task(pid *, pid_type type)
380
381 struct task_struct *pid_task(struct pid *pid, enum pid_type type)
382 {
383     struct task_struct *result = NULL;
384     if (pid) {
385         struct hlist_node *first;
386         first = rcu_dereference_check(hlist_first_rcu(&pid->tasks[type]),
387                                       lockdep_tasklist_lock_is_held());
388         if (first)
389             result = hlist_entry(first, struct task_struct, pid_links[(type)]);
390     }
391     return result;
392 }
393 EXPORT_SYMBOL(pid_task);
394
```

三、pid_nr(...)函数功能：获取进程的全局进程号，具体 Linux 内核源码对应函数设计如下：

```
include > linux > C pid.h > ...
161 /*
162  * the helpers to get the pid's id seen from different namespaces
163  *
164  * pid_nr()    : global id, i.e. the id seen from the init namespace;
165  * pid_vnr()   : virtual id, i.e. the id seen from the pid namespace of
166  *               current.
167  * pid_nr_ns() : id seen from the ns specified.
168  *
169  * see also task_xid_nr() etc in include/linux/sched.h
170  */
171
172 static inline pid_t pid_nr(struct pid *pid)
173 {
174     pid_t nr = 0;
175     if (pid)
176         nr = pid->numbers[0].nr;
177     return nr;
178 }
```

四、__task_pid_nr_ns(...)函数功能：获取进程编号，具体 Linux 内核源码对应函数设计如下：

```
kernel > C pid.c > __task_pid_nr_ns(task_struct *, pid_type type, pid_namespace *)
476
477 pid_t __task_pid_nr_ns(struct task_struct *task, enum pid_type type,
478                        struct pid_namespace *ns)
479 {
480     pid_t nr = 0;
481
482     rcu_read_lock();
483     if (!ns)
484         ns = task_active_pid_ns(current);
485     if (likely(pid_alive(task)))
486         nr = pid_nr_ns(rcu_dereference(*task_pid_ptr(task, type)), ns);
487     rcu_read_unlock();
488
489     return nr;
490 }
491 EXPORT_SYMBOL(__task_pid_nr_ns);
492
```

【运行结果】

```
root@ubuntu: /home/vico/Desktop/pidtest
File  Edit  View  Search  Terminal  Help
onfined" name="/snap/snapd/16292/usr/lib/snapd/snap-confine//mount-namespace-ca
pture-helper" pid=4194 comm="apparmor_parser"
[   358.057034] audit: type=1400 audit(1658819525.995:108): apparmor="STATUS" op
eration="profile_replace" profile="unconfined" name="snap-update-ns.gnome-syste
m-monitor" pid=4196 comm="apparmor_parser"
[   358.870131] audit: type=1400 audit(1658819526.807:109): apparmor="STATUS" op
eration="profile_replace" profile="unconfined" name="snap.gnome-system-monitor.
gnome-system-monitor" pid=4197 comm="apparmor_parser"
[   359.058126] audit: type=1400 audit(1658819526.995:110): apparmor="STATUS" op
eration="profile_replace" profile="unconfined" name="snap.gnome-system-monitor.
hook.configure" pid=4198 comm="apparmor_parser"
[ 3265.903749] pidtest: loading out-of-tree module taints kernel.
[ 3265.903803] pidtest: module verification failed: signature and/or required k
ey missing - tainting kernel
[ 3265.903961] 调用pidtest_initfunc(...)函数.
[ 3265.903963] 打印进程描述符count的值: 3
[ 3265.903963] 打印进程描述符level的值: 0
[ 3265.903963] 打印进程描述符nnumbers的值：5837
[ 3265.903964] 打印进程描述符的全局进程编号为：5837
[ 3265.903964] 打印进程描述符的当前线程组编号为：5837
[ 3265.903964] 打印任务当前的状态为: 0
[ 3265.903965] 打印任务当前的进程号为：5837
[ 3265.903965] 调用__task_pid_nr_ns(...)函数，输出结果为：5837
[ 3265.903965] 退出pidtest_initfunc(...)函数.
root@ubuntu:/home/vico/Desktop/pidtest# rmmod pidtest.ko
root@ubuntu:/home/vico/Desktop/pidtest# dmesg -c
[ 3313.235444] 正常退出Linux内核.....
root@ubuntu:/home/vico/Desktop/pidtest#
```

【程序源码】

```
#include <linux/sched.h>

#include <linux/pid.h>

#include <linux/module.h>


static int __init pidtest_initfunc(void)

{
```

```
    printk("调用 pidtest_initfunc(...)函数.\n");

    // 1 : find_get_pid(...)
    struct pid *kernelpid=find_get_pid(current->pid);
    printk("打印进程描述符 count 的值: %d\n",kernelpid->count);
    printk("打印进程描述符 level 的值: %d\n",kernelpid->level);
    printk("打印进程描述符 nnumbers 的值：%d\n",kernelpid->numbers[kernelpid->level].nr);

    // 2 : pid_nr(..)
    int iNr=pid_nr(kernelpid);
    printk("打印进程描述符的全局进程编号为：%d\n",iNr);
    printk("打印进程描述符的当前线程组编号为：%d\n",current->tgid);

    // 3 : pid_task(...)
    struct task_struct *ttask=pid_task(kernelpid,PIDTYPE_PID);
    printk("打印任务当前的状态为: %ld\n",ttask->state);
    printk("打印任务当前的进程号为：%d\n",ttask->pid);

    // 4 : __task_pid_nr_ns(...)
    pid_t rest=__task_pid_nr_ns(ttask,PIDTYPE_PID,kernelpid->numbers[kernelpid->level].ns);
    printk("调用__task_pid_nr_ns(...)函数，输出结果为：%d\n",rest);

    printk("退出 pidtest_initfunc(...)函数.\n");

    return 0;
}

static void __exit pidtest_exitfunc(void)
{
    printk("正常退出 Linux 内核.....\n");
```

```
}


MODULE_LICENSE("GPL");


module_init(pidtest_initfunc);
module_exit(pidtest_exitfunc);
```