Shanghai Jiao Tong University

CS353 Linux Kernel

Project 2B: Process Management

Chao Gao

5142029014

April 5, 2017

1 Introduction

Recently we have learned something about process management. Now I have a brief understanding about linux process descriptor, process ID and how processes are organized.
Let's recap the requirement of this project.
Process: schedule in times.
Add ctx, a new member to task_struct to record the schedule in times of the process. When a task is scheduled in to run on a CPU.
Export ctx under /proc/XXX/ctx.

2 Environment

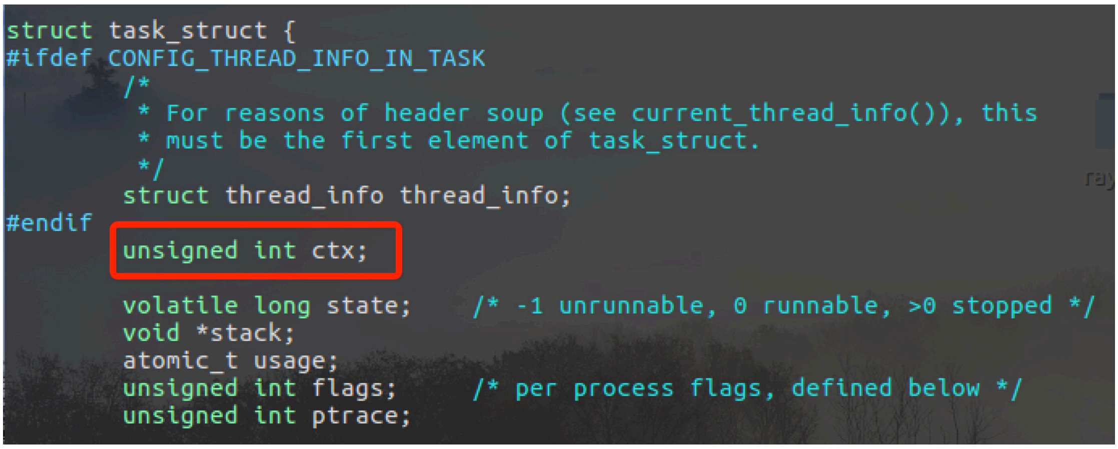
Virtual OS: Ubuntu 16.04.
kernel version: 4.9.13.

3 Overall procedure

Here I will present the whole process of this porject.

3.1 add ctx to task_struct

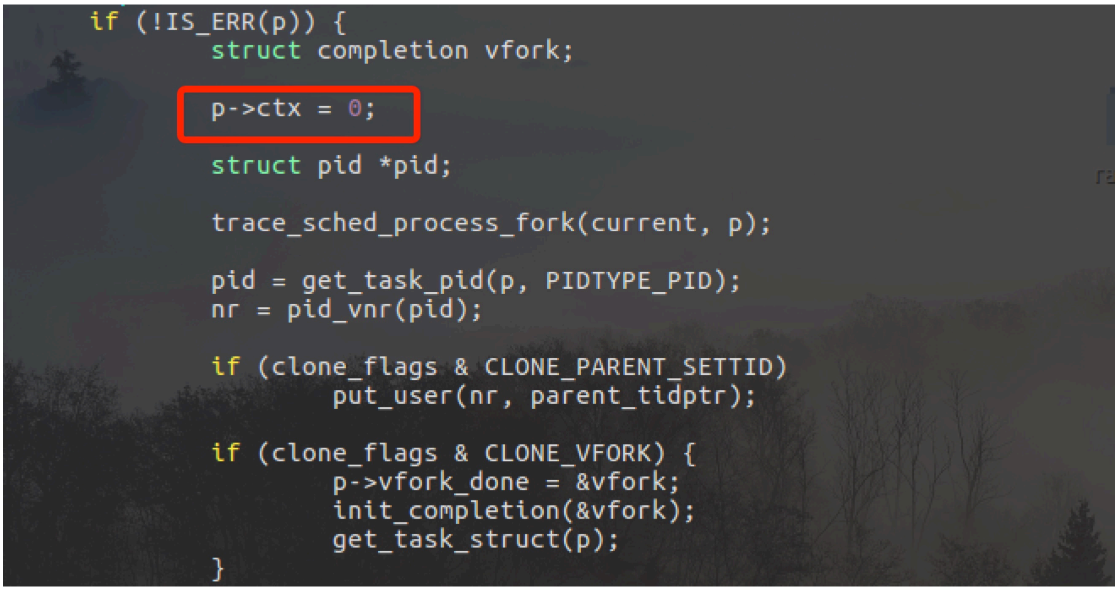
task_struct is in **include/linux/sched.h**.



```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info thread_info;
#endif
        unsigned int ctx;

        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;
```

3.2 ctx initialization

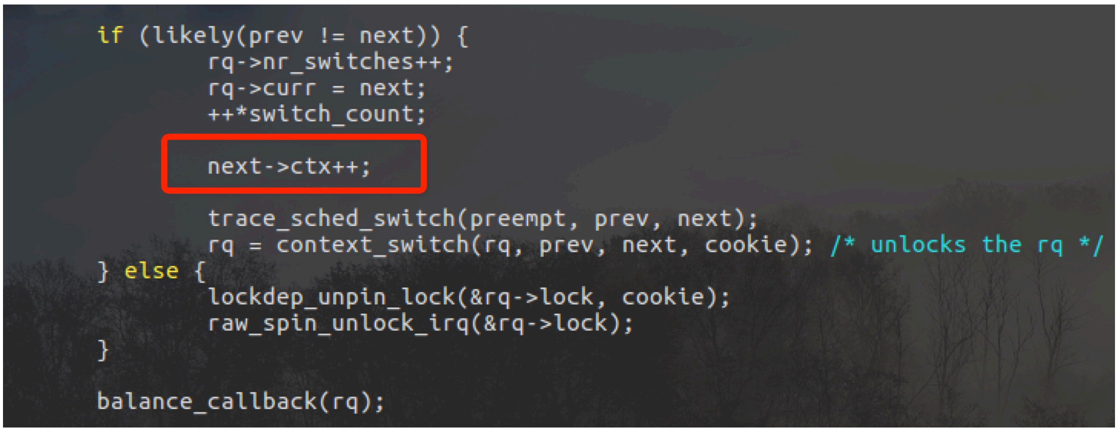
Initialize ctx=0 in function _do_fork in **kernel/fock.c**.



```
	if (!IS_ERR(p)) {
		struct completion vfork;

		p->ctx = 0;

		struct pid *pid;

		trace_sched_process_fork(current, p);

		pid = get_task_pid(p, PIDTYPE_PID);
		nr = pid_vnr(pid);

		if (clone_flags & CLONE_PARENT_SETTID)
			put_user(nr, parent_tidptr);

		if (clone_flags & CLONE_VFORK) {
			p->vfork_done = &vfork;
			init_completion(&vfork);
			get_task_struct(p);
		}
```

3.3 plus ctx

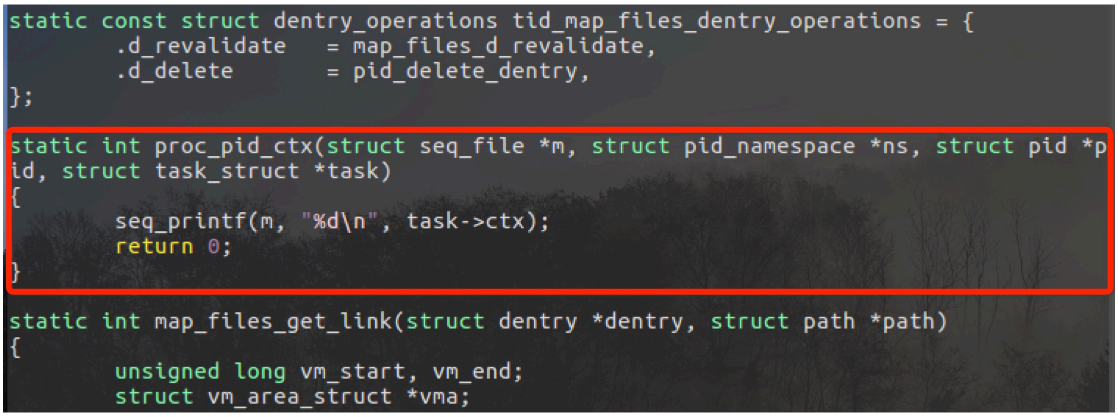
When the processes are switched, plus ctx by 1 in **kernel/sched/core.c**.



```
	if (likely(prev != next)) {
		rq->nr_switches++;
		rq->curr = next;
		++*switch_count;

		next->ctx++;

		trace_sched_switch(preempt, prev, next);
		rq = context_switch(rq, prev, next, cookie); /* unlocks the rq */
	} else {
		lockdep_unpin_lock(&rq->lock, cookie);
		raw_spin_unlock_irq(&rq->lock);
	}

	balance_callback(rq);
```

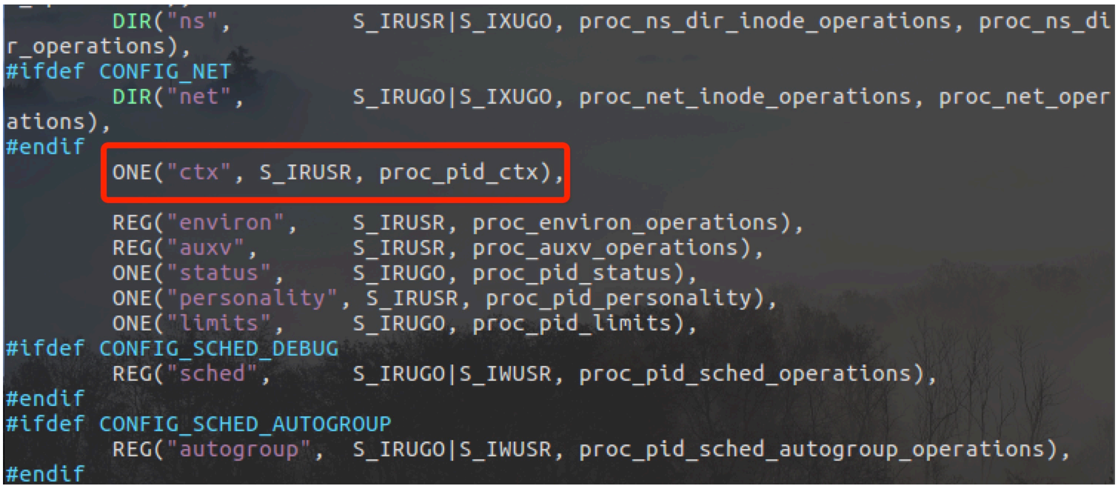
3.4 export ctx

Define a function **proc_pid_ctx** in **/fs/proc/base.c**



```
static const struct dentry_operations tid_map_files_dentry_operations = {
        .d_revalidate   = map_files_d_revalidate,
        .d_delete       = pid_delete_dentry,
};

static int proc_pid_ctx(struct seq_file *m, struct pid_namespace *ns, struct pid *p
id, struct task_struct *task)
{
        seq_printf(m, "%d\n", task->ctx);
        return 0;
}

static int map_files_get_link(struct dentry *dentry, struct path *path)
{
        unsigned long vm_start, vm_end;
        struct vm_area_struct *vma;
```

Create a new entry in the **tgid_base_stuff**[].



```
        DIR("ns",         S_IRUSR|S_IXUGO, proc_ns_dir_inode_operations, proc_ns_di
r_operations),
#ifdef CONFIG_NET
        DIR("net",        S_IRUGO|S_IXUGO, proc_net_inode_operations, proc_net_oper
ations),
#endif
        ONE("ctx", S_IRUSR, proc_pid_ctx),

        REG("environ",    S_IRUSR, proc_environ_operations),
        REG("auxv",       S_IRUSR, proc_auxv_operations),
        ONE("status",     S_IRUGO, proc_pid_status),
        ONE("personality", S_IRUSR, proc_pid_personality),
        ONE("limits",     S_IRUGO, proc_pid_limits),
#ifdef CONFIG_SCHED_DEBUG
        REG("sched",      S_IRUGO|S_IWUSR, proc_pid_sched_operations),
#endif
#ifdef CONFIG_SCHED_AUTOGROUP
        REG("autogroup",  S_IRUGO|S_IWUSR, proc_pid_sched_autogroup_operations),
#endif
```

3.5 compile the kernel

```
make -j4
make modules_install
make install
```

3.6 check result

Write the test program file **test.c**.

```
#include<stdio.h>
 int main()
 {
       while(1)  getchar();
       return 0;
 }
```

Compile and run.

```
gcc test.c -o test -Wall
./block
```

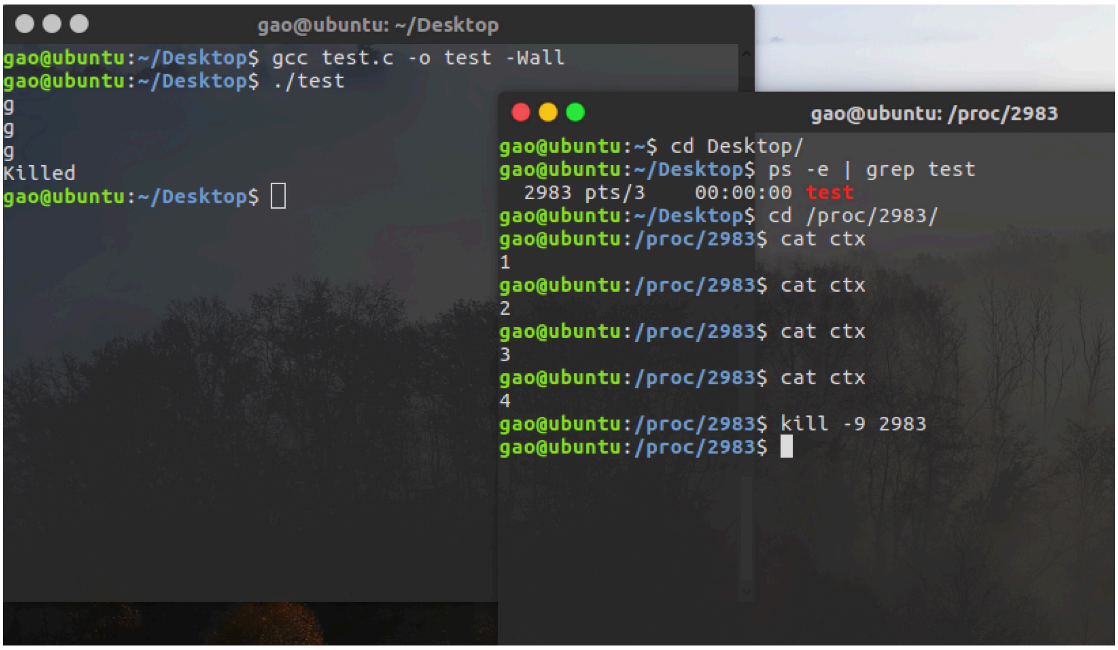
Open another terminal.

```
ps -e | grep test
```

And I get the pid is 2983. Continue to execute

```
cd /proc/2983
cat ctx
```

Back to the original terminal, input any character, and everytime ctx increase by 1. We can see the result.



```
gao@ubuntu:~/Desktop$ gcc test.c -o test -Wall
gao@ubuntu:~/Desktop$ ./test
g
g
g
Killed
gao@ubuntu:~/Desktop$ 
```

```
gao@ubuntu:~$ cd Desktop/
gao@ubuntu:~/Desktop$ ps -e | grep test
  2983 pts/3    00:00:00 test
gao@ubuntu:~/Desktop$ cd /proc/2983/
gao@ubuntu:/proc/2983$ cat ctx
1
gao@ubuntu:/proc/2983$ cat ctx
2
gao@ubuntu:/proc/2983$ cat ctx
3
gao@ubuntu:/proc/2983$ cat ctx
4
gao@ubuntu:/proc/2983$ kill -9 2983
gao@ubuntu:/proc/2983$ 
```

We can see the project has successfully been done.

4 Analysis

Due to detailed handbook and some tutorials on the Internet, I don't meet much challenges. This time I succeed after compiling the kernel only once.
This project is very clear to be divided into two parts. Use ctx to listen to the schedule of a task and export ctx under proc file. However, the source code is very long, it can be tough to read it and insert my code piece.

Due to time limitation, I have not finished the optional part yet. And I will submit my code and report file about the optional part in next assignment submission.

5 Acknowledgement

Thanks a lot to Prof.Chen for impressive teaching about process management and TAs for answering questions..