Linux内核 Lab02 - 进程管理

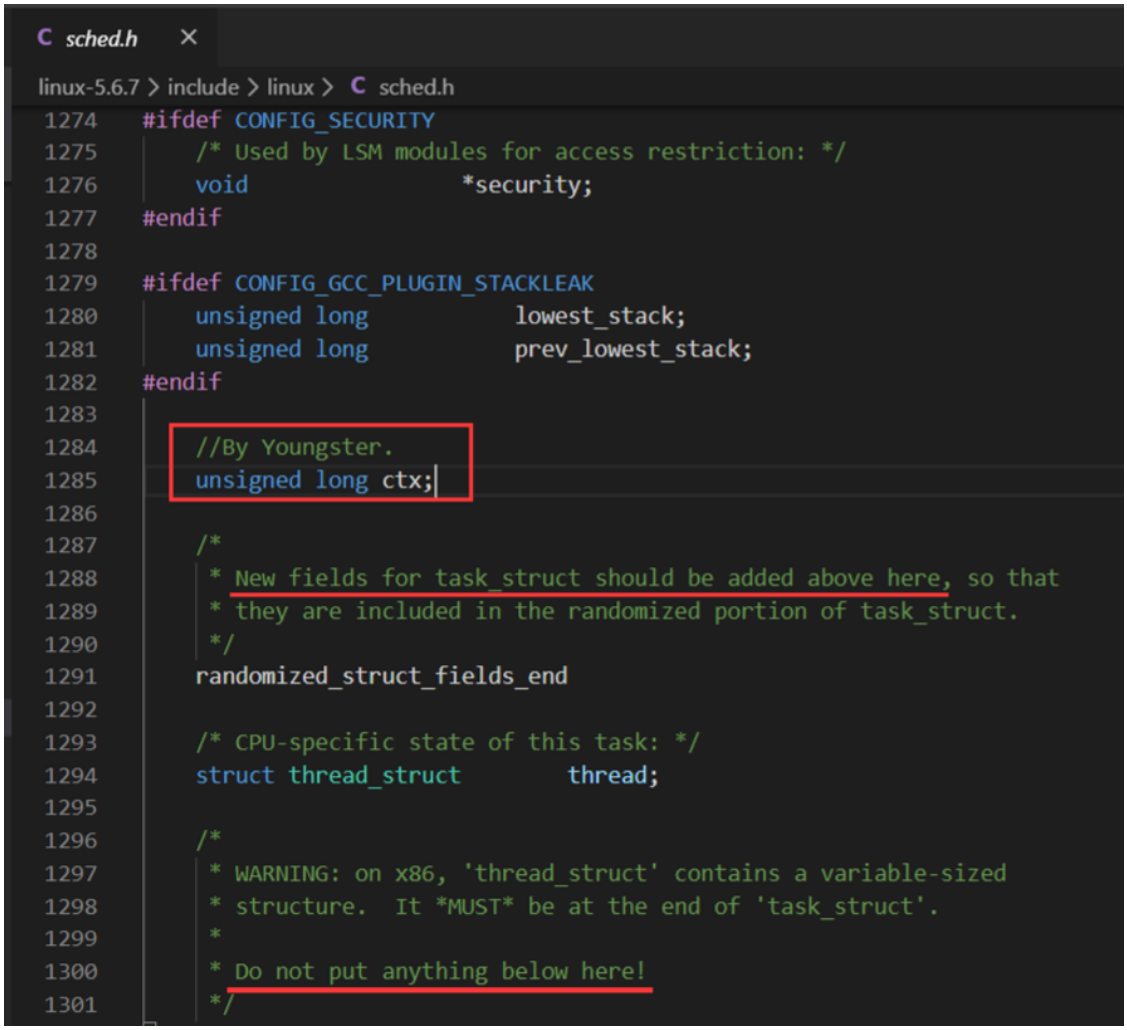
实验内容

1. 为`task_struct`结构添加数据成员`int ctx`，每当进程被调用一次，ctx++。
2. 把ctx输出到`/proc/<pid>/ctx`下，通过`cat /proc/<pid>/ctx`可以查看当前指定进程的ctx的值。

实验过程

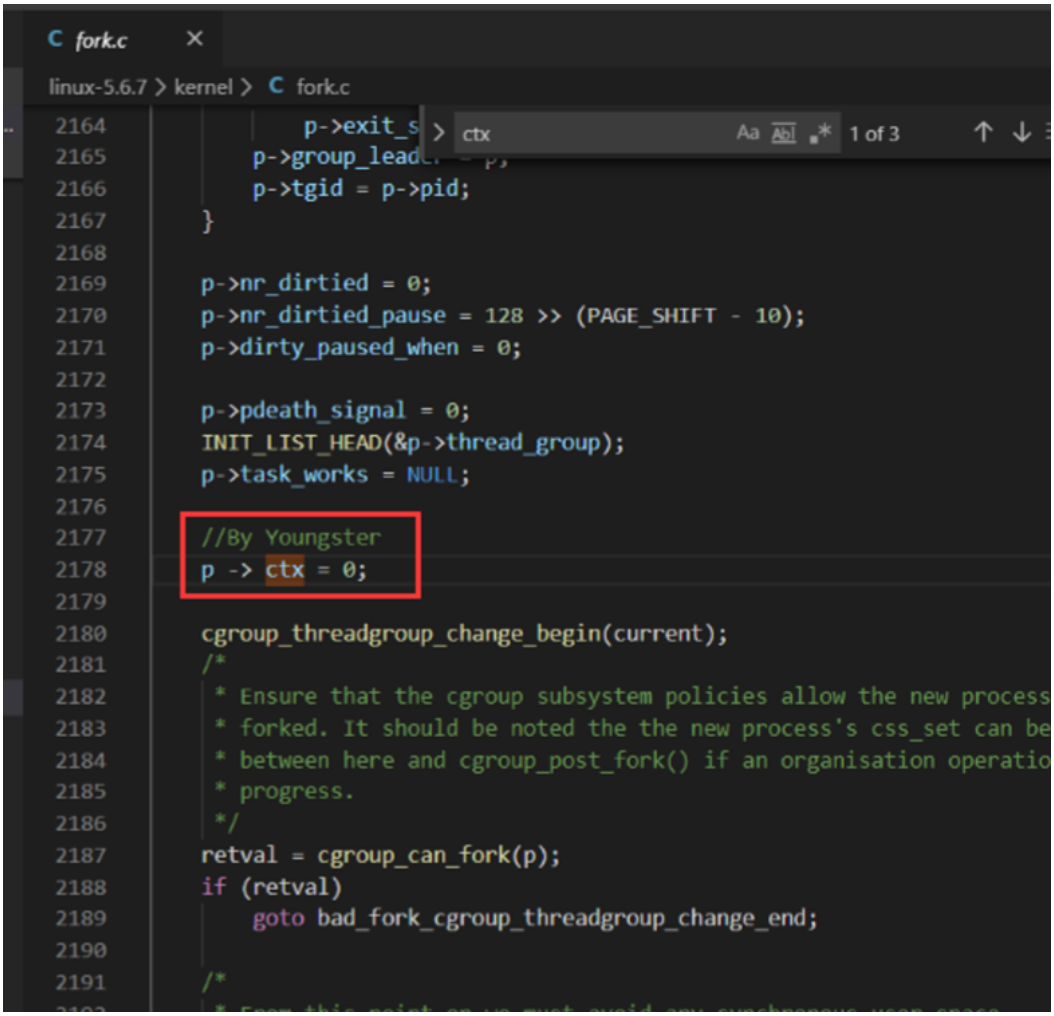
1. 首先在 task_struct 数据结构中添加成员 ctx，原代码注释中有提示新数据成员的位置

```
struct task_struct {
    //original code
    ...


    //my code
    unsigned long ctx;

    //original code

    /*
     * New fields for task_struct should be added above here, so that
     * they are included in the randomized portion of task_struct.
     */
    randomized_struct_fields_end

    /* CPU-specific state of this task: */
    struct thread_struct        thread;

    /*
     * WARNING: on x86, 'thread_struct' contains a variable-sized
     * structure.  It *MUST* be at the end of 'task_struct'.
     *
     * Do not put anything below here!
     */
};
```



2. 然后在fork.c的copy_process中增加初始化参数

```
static __latent_entropy struct task_struct *copy_process(
                    struct pid *pid,
                    int trace,
                    int node,
                    struct kernel_clone_args *args)
{
  // original code
  ...
    /* ok, now we should be set up.. */
  p->pid = pid_nr(pid);
  ...
  // my code
  p -> ctx = 0;

  // original code
  ...

};
```



3. 在core.c中改变ctx的值

```
static void __sched notrace __schedule(bool preempt)
{
    //original code
    struct task_struct *prev, *next;
    unsigned long *switch_count;
    struct rq_flags rf;
    struct rq *rq;
    int cpu;

    ...

    next = pick_next_task(rq, prev, &rf);
    clear_tsk_need_resched(prev);
    clear_preempt_need_resched();

    if (likely(prev != next)) {
        rq->nr_switches++;

        // my code
        if(next != NULL) {
            next -> ctx ++;
        }

        // original code
        ...
```

```
    }
    ...
}
```

core.c

linux-5.6.7 > kernel > sched > core.c

```
4046        switch_count = &prev->nvcsw;
4047    }
4048
4049    next = pick_next_task(rq, prev, &rf);
4050    clear_tsk_need_resched(prev);
4051    clear_preempt_need_resched();
4052
4053    if (likely(prev != next)) {
4054        rq->nr_switches++;
4055
4056        //By Youngster.
4057        if(next != NULL) {
4058            next -> ctx ++;
4059        }
4060
4061        /*
4062         * RCU users of rcu_dereference(rq->curr) may not see
4063         * changes to task_struct made by pick_next_task().
4064         */
4065        RCU_INIT_POINTER(rq->curr, next);
4066        /*
4067         * The membarrier system call requires each architecture
```

4. 在/proc文件系统中增加每个进程对应文件夹的文件,以及对应的回调函数

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

...

static const struct pid_entry tgid_base_stuff[] = {
  // original code
  ...


  // my code
  ONE("ctx", S_IRUSR, proc_pid_ctx),
};
```

base.c

linux-5.6.7 > fs > proc > base.c

```
3087        unsigned long depth = THREAD_SIZE -
3088                    (task->lowest_stack & (THREAD_SIZE - 1));
3089
3090        seq_printf(m, "previous stack depth: %lu\nstack depth: %lu\n",
3091                               prev_depth, depth);
3092        return 0;
3093    }
3094    #endif /* CONFIG_STACKLEAK_METRICS */
3095
3096
3097    //By Youngster
3098    static int proc_pid_ctx(struct seq_file *m, struct pid_namespace *ns,
3099     struct pid *pid, struct task_struct *task)
3100    {
3101        seq_printf(m, "%s %d\n", "ctx: ", task->ctx);
3102        return 0;
3103    }
3104
3105
```

base.c

linux-5.6.7 > fs > proc > base.c

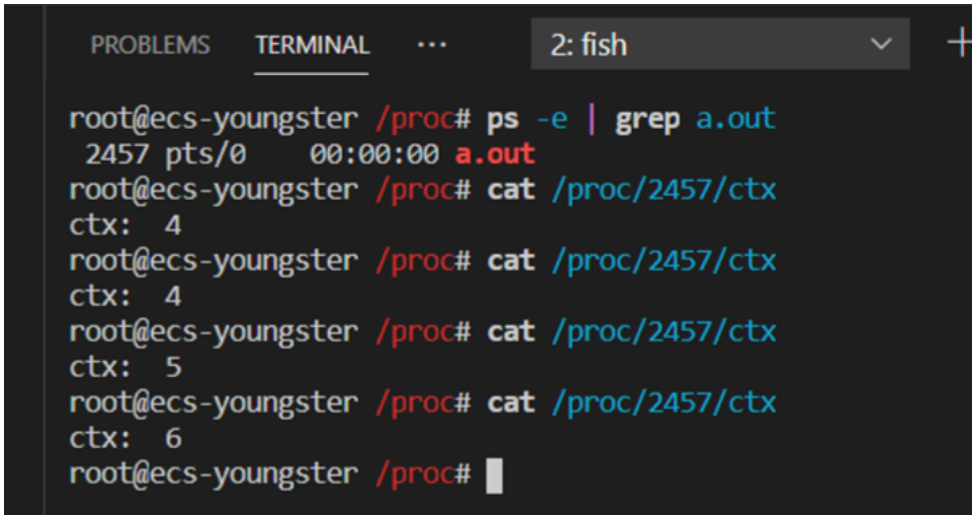
```
3209        REG("timerslack_ns", S_IRUGO|S_IWUGO, proc_pid_set_timerslack_ns_operat
3210    #ifdef CONFIG_LIVEPATCH
3211        ONE("patch_state",  S_IRUSR, proc_pid_patch_state),
3212    #endif
3213    #ifdef CONFIG_STACKLEAK_METRICS
3214        ONE("stack_depth", S_IRUGO, proc_stack_depth),
3215    #endif
3216    #ifdef CONFIG_PROC_PID_ARCH_STATUS
3217        ONE("arch_status", S_IRUGO, proc_pid_arch_status),
3218    #endif
3219        //By Youngster.
3220        ONE("ctx", S_IRUSR, proc_pid_ctx),
3221    };
3222
3223    static int proc_tgid_base_readdir(struct file *file, struct dir_context *ct
3224    {
```

实验效果

在每次接受到字符后，程序都被调度一次，ctx的值增加1

PROBLEMS TERMINAL ··· 1: a.out

```
root@ecs-youngster /h/s/lib02# gcc test.c
root@ecs-youngster /h/s/lib02# ./a.out
1
2
3
4
5
```



实验心得

1. Google比百度好多了!
2. 学会了使用远程服务器，配上vscode的ssh插件以后，和本地系统几乎没太大差别
3. 内核代码编译时间长，写代码需要谨慎，避免不必要的编译/运行时错误

参考:

fork.c中如何初始化进程描述符

https://stackoverflow.com/questions/40949954/linux-kernel-where-is-the-task-struct-process-initialization

core.c中调度进程的函数

https://stackoverflow.com/questions/15608466/entry-and-exit-of-kernel-schedule-function

为/proc/pid 添加文件:

https://lists.kernelnewbies.org/pipermail/kernelnewbies/2011-January/000475.html