Lab3-netlock

qingyunqu

Part I:

- 在 `arch/x86/syscalls/syscall_64.tbl` 中添加三个新的系统调用号：

```
322     64      net_lock                sys_net_lock
323     64      net_unlock              sys_net_unlock
324     64      net_lock_wait_timeout   sys_net_lock_wait_timeout
```

- 在 `inlcude/linux/syscalls.h` 中添加系统调用声明：

```
asmlinkage long sys_net_lock(netlock_t type,u_int16_t timeout_val);
asmlinkage long sys_net_unlock();
asmlinkage long sys_net_lock_wait_timeout();
```

- 在 `kernel/netlock.c` 中实现这三个系统调用
- 声明定义一个读写信号量 `DECLARE_RWSEM(netlock)`
- 在`SYS_net_lock `中初始化网络用户进程的EDF相关数据并改变其调度类，然后获取读写锁：

```
if(type==NET_LOCK_USE){
                init_netlock_use((unsigned long)timeout_val);
                down_read(&netlock);
        }
        else{
                down_write(&netlock);
        }
```

- 在 `SYS_net_unlock` 中释放读写锁并修改网络用户进程的调度类：

```
if(type==NET_LOCK_USE){
                up_read(&netlock);
                fini_netlock();
        }
        else{
                up_write(&netlock);
        }
```

- 在 `SYS_net_lock_wait_timeout` 的实现中出现了一点小问题，我在 `netlock.c` 中定义的全局变量，无法在调试中在不同的进程中共享，故而现在实现的还是简化版本，还只是简单的睡眠一段时间

Part II:

- 在 `include/uapi/linux/sched.h` 中添加EDF策略调度号： `#define SCHED_EDF 6`
- 在 `kernel/sched/sched.h` 中添加： `extern const struct sched_class edf_sched_class;`
- 在 `kernel/sched` 目录下新建文件 `edf.c` 用于实现EDF调度策略
- 将 `rt.c` 中的 `rt_sched_class->next` 修改为 `&edf_sched_class`
- 在 `edf.c` 中的 `edf_sched_class->next` 设置为 `&fair_sched_class`
- 在 `inlcude/linux/sched.h` 中添加EDF类的调度实例结构体：

```
struct sched_edf_entity {
        struct list_head run_list;
        unsigned long deadline;    //delay on jiffies (jiffies+waittime)
        struct task_struct *p;
};
```

- 在 `task_struct` 中添加 `struct sched_edf_entity edf`
- 在 `kernel/sched/sched.h` 中添加edf_rq调度队列结构体：

```
//edf
struct edf_rq {
        struct list_head head;
        unsigned long edf_nr_running;
```


© Copyright Thursday, Nov 30, 2017, 1:12 AM by LiuYuanqiang

```
        //struct sched_edf_entity *curr;
        //unsigned long nearest;   //the nearest deadline,also delay on jiffies
};
```

- 在 `struct rq` 中添加 `struct edf_rq edf`
- 在 `edf.c` 中添加 `init_edf_rq()` 函数，用来初始化 `edf_rq` 队列,同时在 `kernel/sched/core.c` 中的 `sched_init()` 中添加对应的初始化调用：

```
void init_edf_rq(struct edf_rq *edf)
{
        INIT_LIST_HEAD(&edf->head);
        edf->edf_nr_running=0;
        edf->nearest=(unsigned long)(0-1);
}
```

- 在 `edf.c` 中实现 `edf_sched_class` 调度类中的 `dequeue_task_edf` 、 `enqueue_task_edf` 、 `pick_next_task_edf`
- `enqueue_task_of` 中主要是将调度实例加入队列并且设置 `edf` 调度队列的最近deadline值(nearest)：

```
static void enqueue_task_edf(struct rq *rq, struct task_struct *p, int flags)
{
        struct edf_rq *edf_rq=&rq->edf;
        struct sched_edf_entity *edf=&p->edf;
        list_add(&edf->run_list,&edf_rq->head);
        if(edf_rq->nearest > edf->deadline)
                edf_rq->nearest = edf->deadline;
        edf_rq->edf_nr_running++;
}
```

- `dequeue_task_of` 中主要是将调度实例移除队列并且重新计算最近的 `nearest` 值：

```
static void dequeue_task_edf(struct rq *rq, struct task_struct *p, int flags)
{
        list_del(&p->edf.run_list);
        rq->edf.edf_nr_running--;
        if(!list_empty(&rq->edf.head)){
                struct list_head *pos;
                struct list_head *head=&(rq->edf.head);
                struct sched_edf_entity *edf;
                edf=list_first_entry(head,struct sched_edf_entity,run_list);
                rq->edf.nearest=edf->deadline;
                list_for_each(pos,head){
                        edf=list_entry(pos,struct sched_edf_entity,run_list);
                        if(edf->deadline < rq->edf.nearest)
                                rq->edf.nearest = edf->deadline;
                }
        }
        else
                rq->edf.nearest=(unsigned long)(0-1);
}
```

- `pick_next_task_edf` 中通过比较队列的 `nearest` 值和每个进程的 `deadline` 来确定接下来要调度的进程
- 最后是实现 `netlock.c` 中调用的 `init_netlock 和 fini_netlock`，分别是初始化调度实例的各种值并修改其调度类和修改回原来的调度类

Part III:

- 实现了用户空间的两类进程，一个是网络控制器 `sleeper.c` ，另一个是网络用户 `netuser.c` ，具体实现详见代码

© Copyright Thursday, Nov 30, 2017, 1:12 AM by LiuYuanqiang