Nicholas Jordan, Bryce Holley

April 14, 2014

CS 411 Project 1

Our Solution

For implementing the two process scheduling algorithms, Round Robin and FIFO, into the supplied kernel, a stock Linux 3.0.4 kernel found at `https://www.kernel.org/pub/linux/kernel/v3.0` already contains all relevant functionality, specifically within the sched.c and sched_rt.c files in the kernel directory.

By using the diff command with the project kernel and stock 3.0.4 kernel, we were able to identify each of the different additions to be made within these two files in the project kernel. Each addition makes use of other existing functions, variables, and structures already defined elsewhere in the scheduler header file and the two .c files.

The additions are as follows:

sched.c

(1)
```
if (unlikely(policy == SCHED_FIFO || policy == SCHED_RR))
        return 1;
```

This allows the rt_policy function to return 1 if the policy is RR or FIFO, indicating to the caller that the policy is a real-time policy. The unlikely() function lets the compiler know that this branch will usually not be taken, in order to improve performance.

(2)
```
sched_setscheduler_nocheck(stop, SCHED_FIFO, &param);
```

Located within the sched_set_stop_task function, this calls several nested functions to change to FIFO policy.

(3)
```
if (p->policy == SCHED_FIFO || p->policy == SCHED_RR) {
        p->policy = SCHED_NORMAL;
        p->normal_prio = p->static_prio;
        }
```

This allows the sched_fork function to revert to the NORMAL scheduler in the event that the current policy is RR or FIFO, as needed.

(4)
```
if (policy != SCHED_FIFO && policy != SCHED_RR &&
        policy != SCHED_NORMAL && policy != SCHED_BATCH &&
        policy != SCHED_IDLE)
        return -EINVAL;
```

This if-block ensures that the policy passed into the __sched_setscheduler function is a valid scheduler policy (EINVAL => Invalid Argument).

(5)
```
switch (policy) {
case SCHED_FIFO:
case SCHED_RR:
        ret = MAX_USER_RT_PRIO-1;
        break;
...
```

This case is part of a SYSCALL_DEFINE1 that returns a value for sched_get_priority_max. If the scheduling policy is FIFO or RR, the max priority is returned (generally 99).

(6)
```
switch (policy) {
case SCHED_FIFO:
case SCHED_RR:
        ret = 1;
        break;
...
```

Another SYSCALL_DEFINE1, this time for sched_get_priority_min. 1 is returned for FIFO and RR, 0 otherwise.

sched_rt.c

(7)
```
if (p->policy != SCHED_RR)
        return;

if (--p->rt.time_slice)
        return;

p->rt.time_slice = DEF_TIMESLICE;
```

This section sets up the time slice for the given task_struct object. In the case that FIFO is active, time_slice doesnt matter (processes are run whole), and if the time_slice does not fall below 0 after decrementing 1, it continues to set DEF_TIMESLICE for RR policies only.

(8)
```
if (task->policy == SCHED_RR)
        return DEF_TIMESLICE;
else
        return 0;
```

This completes the functionality of get_rr_interval_rt for real-time policies, returning the current timeslice for RR policies, and 0 for FIFO policies.