Looking forward on Proxy Execution

Valentin Schneider <valentin.schneider@arm.com>

25/08/2020

Outline

- Proxy exec TL;DR
- Status update
- Prickly points

Recap

- Why do I care?
  - Priority inheritance++
  - big.LITTLE problems[1]

- rt_mutex
  - Dequeue task when waiting on lock
  - Directly tweak lock owner's `priority` / `sched_class`

- Broken for e.g. deadline tasks

- proxy exec (PE)
  - Do not dequeue task when waiting on lock
  - `pick_next_task()` can still pick it
  - find a task (owner) that can unblock it instead
  - Run owner with waiter's scheduling context (scheduling decisions)
  - Honour owner's execution context (CPU affinity)

+ Relies on the existing scheduler for inheriting properties.

- Need to aggregate dependency chain on a single RQ

[1]: https://lwn.net/Articles/820575/

Status

- Latest update from Juri[1] survives mutex locktorture with `maxcpus=2`
  - Still dies for > 2 CPUs :(
- Rebased onto v5.8-rc4[2]
- Dug into issues
  - Broken with `CONFIG_FAIR_GROUP_SCHED=y` and > 2 CPUs
  - Survives locktorture on `CONFIG_FAIR_GROUP_SCHED=n`
- Plastering here and there
- Plan for now
  - Iron out PE with `CONFIG_FAIR_GROUP_SCHED=n`
  - Re-evaluate CFS screwups then

[1]: https://github.com/jlelli/linux/tree/experimental/deadline/proxy-rfc-v2-debug
[2]: http://www.linux-arm.org/git?p=linux-vs.git;a=shortlog;h=refs/heads/mainline/sched/proxy-rfc-v3

Testing

- What does ”survive locktorture” really mean for PE?
  - Mutex survival is just one part of it
  - rt_mutex locktorture tests inheritance, but not compatible with PE
- For now, hacky tests with CFS / RT tasks
  - CFS busy-loop owns lock; RT task waits on it
  - Runtime was accounted to owner rather than proxy: no RT throttling!
  - Similar fix for DL (runtime enforcement!)

```
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -995,7 +995,7 @@ static int sched_rt_runtime_exceeded(struct rt_rq *rt_rq)
 static void update_curr_rt(struct rq *rq)
 {
-       struct task_struct *curr = rq->curr;
+       struct task_struct *curr = rq->proxy;
        struct sched_rt_entity *rt_se = &curr->rt;
        u64 delta_exec;
        u64 now;

        if (curr->sched_class != &rt_sched_class)
                return;
```

Userspace reporting

- `pe_owner`: CFS busy loop, owns the lock
- `pe_blocker`: FIFO-50, waits on the lock

Proxy execution:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  120 root     -51   0       0      0      0 D  95.4  0.0  86:15.55 pe_blocker
  119 root      20   0       0      0      0 R   4.6  0.0   4:30.86 pe_owner
```

Equivalent with rtmutex:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  119 root     -51   0       0      0      0 R  94.8  0.0   1:34.15 pe_owner
```


arm

Mutex handoff

- When mutex is released, top-waiter is woken
- Optimistic spinner can come and nab lock
- Now-awake waiter can set `MUTEX_HANDOFF` to force next handoff to top-waiter
- PE enforces `MUTEX_HANDOFF` at every unlock
  - Lets us use `mutex_owner()` (more) reliably
  - Should we be worried wrt optimistic spinning?

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה

Extra: transient migration state

- Dependency chain migration happens one RQ at a time
  - > 2 CPUs bugs: transient migration state?
- Can we actually do better?
  - Direct migration to final RQ involves lots of `rq_lock()` juggling