**SUMMARY**

A concurrency bug in MySQL-6.0-backup branch

**DETAILS**

Some details can also be found at: <https://bugs.mysql.com/bug.php?id=48930>

This bug is due to a data race.

Because lock\_state is updated by two different threads holding \*different\* mutexes, this order of operations is possible:

* The backup thread calls kill\_locking\_thread(). It sets lock\_state= LOCK\_SIGNAL protected by the THR\_LOCK\_caller mutex.
* The locking thread gets the lock and around line 189-197 it sets lock\_state= LOCK\_ACQUIRED and starts waiting. It does this protected by the THR\_LOCK\_thread mutex.

|  |  |
| --- | --- |
| Thread1 (be\_thread.cc) | Thread2 (be\_thread.cc) |
| void Locking\_thread\_st::kill\_locking\_thread()  {  ..  pthread\_mutex\_lock(&THR\_LOCK\_caller);  ..  if (lock\_thd && (lock\_state != LOCK\_DONE) && (lock\_state != LOCK\_SIGNAL))  {  lock\_state= LOCK\_SIGNAL;  pthread\_mutex\_lock(&lock\_thd->LOCK\_thd\_data);  lock\_thd->awake(THD::KILL\_CONNECTION);  pthread\_mutex\_unlock(&lock\_thd->LOCK\_thd\_data);  pthread\_cond\_signal(&COND\_thread\_wait);  }  pthread\_mutex\_unlock(&THR\_LOCK\_caller);  ..  } | pthread\_handler\_t backup\_thread\_for\_locking(void \*arg)  {  ..  THD\_SET\_PROC\_INFO(thd, "waiting for signal");  pthread\_mutex\_lock(&locking\_thd->THR\_LOCK\_thread);  locking\_thd->lock\_state= LOCK\_ACQUIRED;  thd->enter\_cond(&locking\_thd->COND\_thread\_wait,  &locking\_thd->THR\_LOCK\_thread,  "Locking thread: holding table locks");  ..  } |
| 1-lock | 1-lock |