



author Cong Wang <cong.wang@bytedance.com> 2022-11-13 16:51:19 -0800
committer Greg Kroah-Hartman <gregkh@linuxfoundation.org> 2022-11-25 17:42:21 +0100
commit [ce57d6474ae999a3b2d442314087473a646a65c7](#) (patch)
tree [6d46d0586d053e85840a595e98bfe239c0320147](#)
parent [7a704dbfd3735304e261f2787c52fbc7c3884736](#) (diff)
download [linux-ce57d6474ae999a3b2d442314087473a646a65c7.tar.gz](#)

diff options

context: ▼
space: ▼
mode: ▼

kcm: close race conditions on sk_receive_queue

commit 5121197ecc5db58c07da95eb1ff82b98b121a221 upstream.

sk->sk_receive_queue is protected by skb queue lock, but for KCM sockets its RX path takes mux->rx_lock to protect more than just skb queue. However, kcm_recvmmsg() still only grabs the skb queue lock, so race conditions still exist.

We can teach kcm_recvmmsg() to grab mux->rx_lock too but this would introduce a potential performance regression as struct kcm_mux can be shared by multiple KCM sockets.

So we have to enforce skb queue lock in requeue_rx_msgs() and handle skb peek case carefully in kcm_wait_data(). Fortunately, skb_recv_datagram() already handles it nicely and is widely used by other sockets, we can just switch to skb_recv_datagram() after getting rid of the unnecessary sock lock in kcm_recvmmsg() and kcm_splice_read(). Side note: SOCK_DONE is not used by KCM sockets, so it is safe to get rid of this check too.

I ran the original syzbot reproducer for 30 min without seeing any issue.

Fixes: [ab7ac4eb9832](#) ("kcm: Kernel Connection Multiplexor module")
Reported-by: syzbot+278279efdd2730dd14bf@syzkaller.appspotmail.com
Reported-by: shaozhengchao <shaozhengchao@huawei.com>
Cc: Paolo Abeni <pabeni@redhat.com>
Cc: Tom Herbert <tom@herbertland.com>
Signed-off-by: Cong Wang <cong.wang@bytedance.com>
Link: <https://lore.kernel.org/r/20221114005119.597905-1-xiyong.wangcong@gmail.com>
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Diffstat

```
-rw-r--r-- net/kcm/kcsock.c 60
```

1 files changed, 8 insertions, 52 deletions

diff --git a/net/kcm/kcsock.c b/net/kcm/kcsock.c

index 0556fdd3644e56..1975403ce8bee2 100644

--- a/net/kcm/kcsock.c

+++ b/net/kcm/kcsock.c

@@ -221,7 +221,7 @@ static void requeue_rx_msgs(struct kcm_mux *mux, struct sk_buff_head *head)

```

    struct sk_buff *skb;
    struct kcm_sock *kcm;

-   while ((skb = __skb_dequeue(head))) {
+   while ((skb = skb_dequeue(head))) {
        /* Reset destructor to avoid calling kcm_rcv_ready */
        skb->destructor = sock_rfree;
        skb_orphan(skb);
@@ -1086,53 +1086,18 @@ out_error:
    return err;
}

-static struct sk_buff *kcm_wait_data(struct sock *sk, int flags,
-                                     long timeo, int *err)
-{
-    struct sk_buff *skb;
-
-    while (!(skb = skb_peek(&sk->sk_receive_queue))) {
-        if (sk->sk_err) {
-            *err = sock_error(sk);
-            return NULL;
-        }
-
-        if (sock_flag(sk, SOCK_DONE))
-            return NULL;
-
-        if ((flags & MSG_DONTWAIT) || !timeo) {
-            *err = -EAGAIN;
-            return NULL;
-        }
-
-        sk_wait_data(sk, &timeo, NULL);
-
-        /* Handle signals */
-        if (signal_pending(current)) {
-            *err = sock_intr_errno(timeo);
-            return NULL;
-        }
-    }
-
-    return skb;
-}

static int kcm_recvmsg(struct socket *sock, struct msghdr *msg,
                      size_t len, int flags)
{
+    int noblock = flags & MSG_DONTWAIT;
    struct sock *sk = sock->sk;
    struct kcm_sock *kcm = kcm_sk(sk);
    int err = 0;
-    long timeo;
    struct strp_msg *stm;
    int copied = 0;
    struct sk_buff *skb;

-    timeo = sock_rcvtimeo(sk, flags & MSG_DONTWAIT);
-
-    lock_sock(sk);
-
-    skb = kcm_wait_data(sk, flags, timeo, &err);
+    skb = skb_rcv_datagram(sk, flags, noblock, &err);
    if (!skb)

```

```

        goto out;

@@ -1163,14 +1128,11 @@ msg_finished:
        /* Finished with message */
        msg->msg_flags |= MSG_EOR;
        KCM_STATS_INCR(kcm->stats.rx_msgs);
-       skb_unlink(skb, &sk->sk_receive_queue);
-       kfree_skb(skb);
    }
}

out:
-       release_sock(sk);
-
+       skb_free_datagram(sk, skb);
    return copied ? : err;
}

@@ -1178,9 +1140,9 @@ static ssize_t kcm_splice_read(struct socket *sock, loff_t *ppos,
                             struct pipe_inode_info *pipe, size_t len,
                             unsigned int flags)
{
+       int noblock = flags & MSG_DONTWAIT;
    struct sock *sk = sock->sk;
    struct kcm_sock *kcm = kcm_sk(sk);
-       long timeo;
    struct strp_msg *stm;
    int err = 0;
    ssize_t copied;
@@ -1188,11 +1150,7 @@ static ssize_t kcm_splice_read(struct socket *sock, loff_t *ppos,

    /* Only support splice for SOCKSEQPACKET */

-       timeo = sock_rcvtimeo(sk, flags & MSG_DONTWAIT);
-
-       lock_sock(sk);
-
-       skb = kcm_wait_data(sk, flags, timeo, &err);
+       skb = skb_recv_datagram(sk, flags, noblock, &err);
    if (!skb)
        goto err_out;

@@ -1220,13 +1178,11 @@ static ssize_t kcm_splice_read(struct socket *sock, loff_t *ppos,
    * finish reading the message.
    */

-       release_sock(sk);
-
+       skb_free_datagram(sk, skb);
    return copied;

err_out:
-       release_sock(sk);
-
+       skb_free_datagram(sk, skb);
    return err;
}

```