커널 네트워크 소스 편집& 컴파일 및 확인

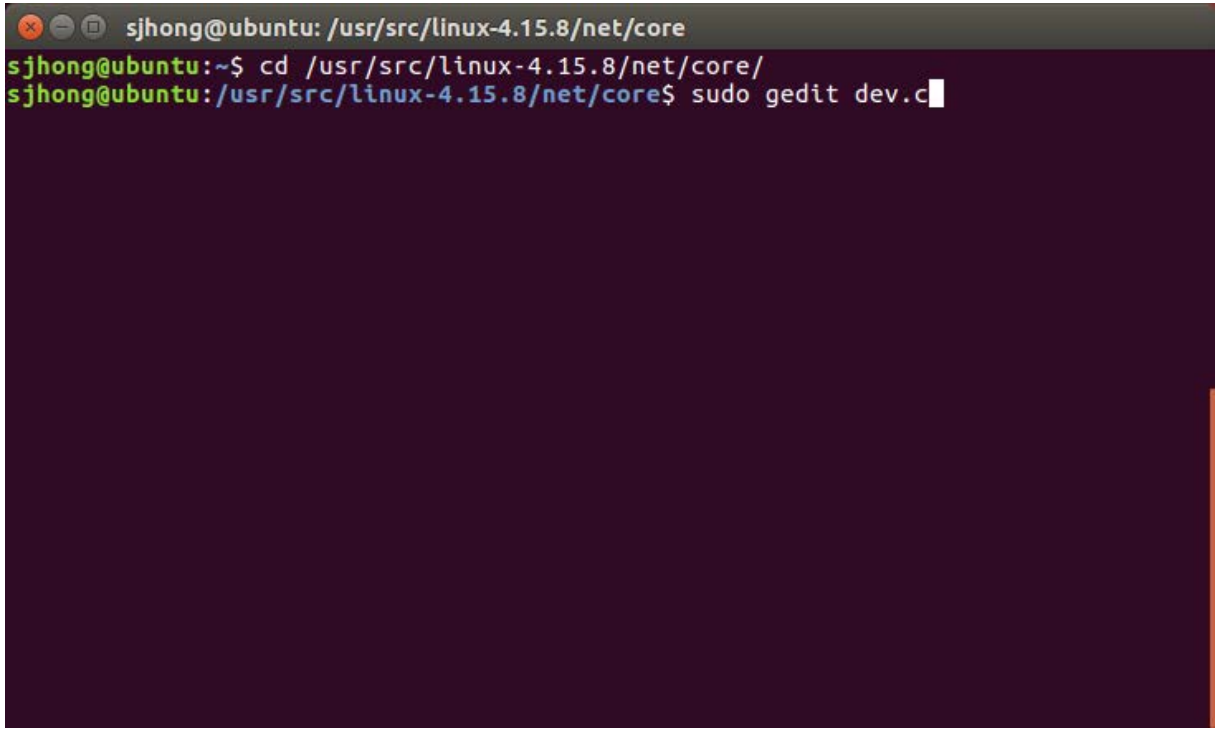
컴퓨터 통신 실습

홍석준

I. 커널 소스 편집

네트워크 커널 소스(layer 2) 편집

- Layer 2 소스 코드인 dev.c를 수정하기 위해 cd /usr/src/linux-4.15.8(자신이 받은 소스 버전)/net/core로 이동
- sudo gedit dev.c로 편집기 open



I. 커널 소스 편집

네트워크 커널 소스(layer 2) 편집

❑ 아래 그림과 같이 printk 문 추가(net_rx_action)

Open dev.c /usr/src/linux-4.15.8/net/core Save

```
        return work;
}

static __latent_entropy void net_rx_action(struct softirq_action *h)
{
        struct softnet_data *sd = this_cpu_ptr(&softnet_data);
        unsigned long time_limit = jiffies +
                usecs_to_jiffies(netdev_budget_usecs);
        int budget = netdev_budget;
        LIST_HEAD(list);
        LIST_HEAD(repoll);

        printk("net_rx_action\n");

        local_irq_disable();
        list_splice_init(&sd->poll_list, &list);
        local_irq_enable();

        for (;;) {
                struct napi_struct *n;

                if (list_empty(&list)) {
                        if (!sd_has_rps_ipi_waiting(sd) && list_empty
(&repoll))
```

C Tab Width: 8 Ln 5688, Col 2 INS

I. 커널 소스 편집

네트워크 커널 소스(layer 2) 편집

❑ 아래 그림과 같이 printk 문 추가(__netif_receive_skb_core)

Open ▾ dev.c /usr/src/linux-4.15.8/net/core Save

```
                rcu_read_unlock();
                return ingress_retval;
        }
#endif /* CONFIG_NETFILTER_INGRESS */
        return 0;
}

static int __netif_receive_skb_core(struct sk_buff *skb, bool pfmemalloc)
{
        struct packet_type *ptype, *pt_prev;
        rx_handler_func_t *rx_handler;
        struct net_device *orig_dev;
        bool deliver_exact = false;
        int ret = NET_RX_DROP;
        __be16 type;

        printk("__netif_receive_skb_core\n");

        net_timestamp_check(!netdev_tstamp_prequeue, skb);

        trace_netif_receive_skb(skb);

        orig_dev = skb->dev;

        skb_reset_network_header(skb);
        if (!skb_transport_header_was_set(skb))
```

C ▾ Tab Width: 8 ▾ Ln 4372, Col 31 ▾ INS

I. 커널 소스 편집

네트워크 커널 소스(layer 2) 편집

❑ 아래 그림과 같이 printk 문 추가(net_tx_action)

Open dev.c /usr/src/linux-4.15.8/net/core Save

```
}
EXPORT_SYMBOL(netif_rx_ni);

static __latent_entropy void net_tx_action(struct softirq_action *h)
{
        struct softnet_data *sd = this_cpu_ptr(&softnet_data);

        printk("net_tx_action\n");

        if (sd->completion_queue) {
                struct sk_buff *clist;

                local_irq_disable();
                clist = sd->completion_queue;
                sd->completion_queue = NULL;
                local_irq_enable();

                while (clist) {
                        struct sk_buff *skb = clist;

                        clist = clist->next;

                        WARN_ON(refcount_read(&skb->users));
                        if (likely(get_kfree_skb_cb(skb)->reason ==
SKB_REASON_CONSUMED))
```

C Tab Width: 8 Ln 4145, Col 1 INS

네트워크 커널 소스(layer 2) 편집

❑ 아래 그림과 같이 printk 문 추가(__dev_queue_xmit)

Open ▾ | dev.c /usr/src/linux-4.15.8/net/core | Save

```
 *      When calling this method, interrupts MUST be enabled.  This is because
 *      the BH enable code must have IRQs enabled so that it will not
deadlock.
 *          --BLG
 */
static int __dev_queue_xmit(struct sk_buff *skb, void *accel_priv)
{
        struct net_device *dev = skb->dev;
        struct netdev_queue *txq;
        struct Qdisc *q;
        int rc = -ENOMEM;

        printk("__dev_queue_xmit\n")

        skb_reset_mac_header(skb);

        if (unlikely(skb_shinfo(skb)->tx_flags & SKBTX_SCHED_TSTAMP))
                __skb_tstamp_tx(skb, NULL, skb->sk, SCM_TSTAMP_SCHED);

        /* Disable soft irqs for various locks below. Also
         * stops preemption for RCU.
         */
        rcu_read_lock_bh();

        skb_update_prio(skb);
```

C ▾ | Tab Width: 8 ▾ | Ln 3441, Col 18 ▾ | INS

I. 커널 소스 편집

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_rcv)

Open ip_input.c /usr/src/linux-4.15.8/net/ipv4 Save

```
}

/*
 *      Main IP Receive routine.
 */
int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type
*pt, struct net_device *orig_dev)
{
        const struct iphdr *iph;
        struct net *net;
        u32 len;

        /* When the interface is in promisc. mode, drop all the crap
         * that it receives, do not try to analyse it.
         */
        if (skb->pkt_type == PACKET_OTHERHOST)
                goto drop;

        printk("ip_rcv\n");

        net = dev_net(dev);
        __IP_UPD_PO_STATS(net, IPSTATS_MIB_IN, skb->len);

        skb = skb_share_check(skb, GFP_ATOMIC);
        if (!skb) {
                __IP_INC_STATS(net, IPSTATS_MIB_INDISCARDS);
```

C Tab Width: 8 Ln 430, Col 9 INS

I. 커널 소스 편집

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_rcv_finish)

Open ip_input.c /usr/src/linux-4.15.8/net/ipv4 Save

```
        return false;
drop:
        return true;
}

static int ip_rcv_finish(struct net *net, struct sock *sk, struct sk_buff
*skb)
{
        const struct iphdr *iph = ip_hdr(skb);
        int (*edemux)(struct sk_buff *skb);
        struct net_device *dev = skb->dev;
        struct rtable *rt;
        int err;

        /* if ingress device is enslaved to an L3 master device pass the
         * skb to its handler for processing
         */
        printk("ip_rcv_finish\n");

        skb = l3mdev_ip_rcv(skb);
        if (!skb)
                return NET_RX_SUCCESS;

        if (net->ipv4.sysctl_ip_early_demux &&
```

C Tab Width: 8 Ln 315, Col 12 INS

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_local_deliver)

Open ip_input.c /usr/src/linux-4.15.8/net/ipv4 Save

```
}

/*
 *      Deliver IP Packets to the higher protocol layers.
 */
int ip_local_deliver(struct sk_buff *skb)
{
        /*
         *      Reassemble IP fragments.
         */
        struct net *net = dev_net(skb->dev);

        printk("ip_local_deliver\n");

        if (ip_is_fragment(ip_hdr(skb))) {
                if (ip_defrag(net, skb, IP_DEFRAG_LOCAL_DELIVER))
                        return 0;
        }

        return NF_HOOK(NFPROTO_IPV4, NF_INET_LOCAL_IN,
                       net, NULL, skb, skb->dev, NULL,
                       ip_local_deliver_finish);
}

static inline bool ip_rcv_options(struct sk_buff *skb)
```

C Tab Width: 8 Ln 247, Col 42 INS

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_local_deliver_finish)

Open ip_input.c /usr/src/linux-4.15.8/net/ipv4 Save

```
                raw_rcv(last, skb);
                return true;
        }
        return false;
}

static int ip_local_deliver_finish(struct net *net, struct sock *sk, struct
sk_buff *skb)
{
        printk("ip_local_deliver_finish\n");

        __skb_pull(skb, skb_network_header_len(skb));

        rcu_read_lock();
        {
                int protocol = ip_hdr(skb)->protocol;
                const struct net_protocol *ipprot;
                int raw;

        resubmit:
                raw = raw_local_deliver(skb, protocol);

                ipprot = rcu_dereference(inet_protos[protocol]);
                if (ipprot) {
                        int ret;
```

C Tab Width: 8 Ln 190, Col 2 INS

I. 커널 소스 편집

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_local_out)

Open | ip_output.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
/* Generate a checksum for an outgoing IP datagram. */
void ip_send_check(struct iphdr *iph)
{
        iph->check = 0;
        iph->check = ip_fast_csum((unsigned char *)iph, iph->ihl);
}
EXPORT_SYMBOL(ip_send_check);

int __ip_local_out(struct net *net, struct sock *sk, struct sk_buff *skb)
{
        struct iphdr *iph = ip_hdr(skb);

        printk("__ip_local_out \n");

        iph->tot_len = htons(skb->len);
        ip_send_check(iph);

        /* if egress device is enslaved to an L3 master device pass the
         * skb to its handler for processing
         */
        skb = l3mdev_ip_out(sk, skb);
        if (unlikely(!skb))
                return 0;

        skb->protocol = htons(ETH_P_IP);

        return nf_hook(NFPROTO_IPV4, NF_INET_LOCAL_OUT,
                       net, sk, skb, NULL, skb_dst(skb)->dev,
                       dst_output);
}
```

C | Tab Width: 8 | Ln 91, Col 2 | INS

I. 커널 소스 편집

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_output)

Open ip_output.c /usr/src/linux-4.15.8/net/ipv4 Save

```
        return NF_HOOK_COND(NFPROTO_IPV4, NF_INET_POST_ROUTING,
                            net, sk, skb, NULL, skb->dev,
                            ip_finish_output,
                            !(IPCB(skb)->flags & IPSKB_REROUTED));
}

int ip_output(struct net *net, struct sock *sk, struct sk_buff *skb)
{
        struct net_device *dev = skb_dst(skb)->dev;

        printk("ip_output\n");

        IP_UPD_PO_STATS(net, IPSTATS_MIB_OUT, skb->len);

        skb->dev = dev;
        skb->protocol = htons(ETH_P_IP);

        return NF_HOOK_COND(NFPROTO_IPV4, NF_INET_POST_ROUTING,
                            net, sk, skb, NULL, dev,
                            ip_finish_output,
                            !(IPCB(skb)->flags & IPSKB_REROUTED));
}

/*
 * copy saddr and daddr, possibly using 64bit load/stores
 * Equivalent to :
 *   iph->saddr = fl4->saddr;
 *   iph->daddr = fl4->daddr;
 */
```

C Tab Width: 8 Ln 406, Col 31 INS

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_finish_output)

Open ip_output.c /usr/src/linux-4.15.8/net/ipv4 Save

```
        } while (segs);

        return ret;
}

static int ip_finish_output(struct net *net, struct sock *sk, struct sk_buff
*skb)
{
        unsigned int mtu;
        int ret;

        printk("ip_finish_output\n");

        ret = BPF_CGROUP_RUN_PROG_INET_EGRESS(sk, skb);
        if (ret) {
                kfree_skb(skb);
                return ret;
        }

#if defined(CONFIG_NETFILTER) && defined(CONFIG_XFRM)
        /* Policy lookup after SNAT yielded a new policy */
        if (skb_dst(skb)->xfrm) {
                IPCB(skb)->flags |= IPSKB_REROUTED;
                return dst_output(net, sk, skb);
        }
#endif
        mtu = ip_skb_dst_mtu(sk, skb);
        if (skb_is_gso(skb))
                return ip_finish_output_gso(net, sk, skb, mtu);
```

C Tab Width: 8 Ln 293, Col 20 INS

네트워크 커널 소스(layer 3) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_finish_output2)

Open ip_output.c /usr/src/linux-4.15.8/net/ipv4 Save

```
                skb->mark = sk->sk_mark;

        /* Send it out. */
        return ip_local_out(net, skb->sk, skb);
}
EXPORT_SYMBOL_GPL(ip_build_and_send_pkt);

static int ip_finish_output2(struct net *net, struct sock *sk, struct sk_buff
*skb)
{
        struct dst_entry *dst = skb_dst(skb);
        struct rtable *rt = (struct rtable *)dst;
        struct net_device *dev = dst->dev;
        unsigned int hh_len = LL_RESERVED_SPACE(dev);
        struct neighbour *neigh;
        u32 nexthop;

        printk("ip finish output2\n");

        if (rt->rt_type == RTN_MULTICAST) {
                IP_UPD_PO_STATS(net, IPSTATS_MIB_OUTMCAST, skb->len);
        } else if (rt->rt_type == RTN_BROADCAST)
                IP_UPD_PO_STATS(net, IPSTATS_MIB_OUTBCAST, skb->len);

        /* Be paranoid, rather than too clever. */
        if (unlikely(skb_headroom(skb) < hh_len && dev->header_ops)) {
                struct sk_buff *skb2;

                skb2 = skb_realloc_headroom(skb, LL_RESERVED_SPACE(dev));
```

C Tab Width: 8 Ln 184, Col 42 INS

I. 커널 소스 편집

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_v4_rcv)

Open ▾ | tcp_ipv4.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
}

/*
 *      From tcp_input.c
 */

int tcp_v4_rcv(struct sk_buff *skb)
{
        struct net *net = dev_net(skb->dev);
        int sdif = inet_sdif(skb);
        const struct iphdr *iph;
        const struct tcphdr *th;
        bool refcounted;
        struct sock *sk;
        int ret;

        printk("tcp_v4_rcv\n");

        if (skb->pkt_type != PACKET_HOST)
                goto discard_it;

        /* Count it even if it's bad */
        __TCP_INC_STATS(net, TCP_MIB_INSEGS);

        if (!pskb_may_pull(skb, sizeof(struct tcphdr)))
                goto discard_it;
```

C ▾ | Tab Width: 8 ▾ | Ln 1628, Col 1 ▾ | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_v4_do_rcv)

Open ▾ | tcp_ipv4.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
 *
 * We have a potential double-lock case here, so even when
 * doing backlog processing we use the BH locking scheme.
 * This is because we cannot sleep with the original spinlock
 * held.
 */
int tcp_v4_do_rcv(struct sock *sk, struct sk_buff *skb)
{
        struct sock *rsk;

        printk("tcp_v4_do_rcv\n");

        if (sk->sk_state == TCP_ESTABLISHED) { /* Fast path */
                struct dst_entry *dst = sk->sk_rx_dst;

                sock_rps_save_rxhash(sk, skb);
                sk_mark_napi_id(sk, skb);
                if (dst) {
                        if (inet_sk(sk)->rx_dst_ifindex != skb->skb_iif ||
                            !dst->ops->check(dst, 0)) {
                                dst_release(dst);
                                sk->sk_rx_dst = NULL;
                        }
                }
                tcp_rcv_established(sk, skb, tcp_hdr(skb));
```

C ▾ | Tab Width: 8 ▾ | Ln 1450, Col 49 ▾ | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_rcv_established)

Open | tcp_input.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
 *      When these conditions are not satisfied it drops into a standard
 *      receive procedure patterned after RFC793 to handle all cases.
 *      The first three cases are guaranteed by proper pred_flags setting,
 *      the rest is checked inline. Fast processing is turned on in
 *      tcp_data_queue when everything is OK.
 */
void tcp_rcv_established(struct sock *sk, struct sk_buff *skb,
                         const struct tcphdr *th)
{
        unsigned int len = skb->len;
        struct tcp_sock *tp = tcp_sk(sk);

        printk("tcp_rcv_established\n");

        tcp_mstamp_refresh(tp);
        if (unlikely(!sk->sk_rx_dst))
                inet_csk(sk)->icsk_af_ops->sk_rx_dst_set(sk, skb);
        /*
         *      Header prediction.
         *      The code loosely follows the one in the famous
         *      "30 instruction TCP receive" Van Jacobson mail.
         *
         *      Van's trick is to deposit buffers into socket queue
         *      on a device interrupt, to call tcp_recv function
         *      on the receive process context and checksum and copy
         *      the buffer to user space. smart...
         *
         *      Our current scheme is not silly either but we take the
```

C | Tab Width: 8 | Ln 5292, Col 34 | INS

I. 커널 소스 편집

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_data_queue)

Open | tcp_input.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
err_free:
        kfree_skb(skb);
err:
        return err;

}

static void tcp_data_queue(struct sock *sk, struct sk_buff *skb)
{
        struct tcp_sock *tp = tcp_sk(sk);
        bool fragstolen;
        int eaten;

        printk("tcp_data_queue\n");

        if (TCP_SKB_CB(skb)->seq == TCP_SKB_CB(skb)->end_seq) {
                __kfree_skb(skb);
                return;
        }
        skb_dst_drop(skb);
        __skb_pull(skb, tcp_hdr(skb)->doff * 4);

        tcp_ecn_accept_cwr(tp, skb);

        tp->rx_opt.dsack = 0;

        /*  Queue data for delivery to the user.
         *  Packets in sequence go to the receive queue.
         *  Out of sequence packets to the out_of_order_queue.
         */
```

C | Tab Width: 8 | Ln 4572, Col 25 | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_recvmsg)

Open | tcp.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
 *      This routine copies from a sock struct into the user buffer.
 *
 *      Technical note: in 2.3 we work on _locked_ socket, so that
 *      tricks with *seq access order and skb->users are not required.
 *      Probably, code can be easily improved even more.
 */

int tcp_recvmsg(struct sock *sk, struct msghdr *msg, size_t len, int nonblock,
                int flags, int *addr_len)
{
        struct tcp_sock *tp = tcp_sk(sk);
        int copied = 0;
        u32 peek_seq;
        u32 *seq;
        unsigned long used;
        int err;
        int target;             /* Read at least this many bytes */
        long timeo;
        struct sk_buff *skb, *last;
        u32 urg_hole = 0;
        struct scm_timestamping tss;
        bool has_tss = false;

        printk("tcp_recvmsg\n");

        if (unlikely(flags & MSG_ERRQUEUE))
                return inet_recv_error(sk, msg, len, addr_len);

        if (sk_can_busy_loop(sk) && skb_queue_empty(&sk->sk_receive_queue) &&
            (sk->sk_state == TCP_ESTABLISHED))
```

C | Tab Width: 8 | Ln 1793, Col 28 | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_sendmsg_locked)

Open | tcp.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
        return err;
}

int tcp_sendmsg_locked(struct sock *sk, struct msghdr *msg, size_t size)
{
        struct tcp_sock *tp = tcp_sk(sk);
        struct ubuf_info *uarg = NULL;
        struct sk_buff *skb;
        struct sockcm_cookie sockc;
        int flags, err, copied = 0;
        int mss_now = 0, size_goal, copied_syn = 0;
        bool process_backlog = false;
        bool sg;
        long timeo;

        printk("tcp sendmsg locked\n");

        flags = msg->msg_flags;

        if (flags & MSG_ZEROCOPY && size) {
                if (sk->sk_state != TCP_ESTABLISHED) {
                        err = -EINVAL;
                        goto out_err;
                }

                skb = tcp_write_queue_tail(sk);
                uarg = sock_zerocopy_realloc(sk, size, skb_zcopy(skb));
                if (!uarg) {
                        err = -ENOBUFS;
                        goto out_err;
```

C | Tab Width: 8 | Ln 1186, Col 42 | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_push)

Open | tcp.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
{
        return skb->len < size_goal &&
               sock_net(sk)->ipv4.sysctl_tcp_autocorking &&
               skb != tcp_write_queue_head(sk) &&
               refcount_read(&sk->sk_wmem_alloc) > skb->truesize;
}

static void tcp_push(struct sock *sk, int flags, int mss_now,
                     int nonagle, int size_goal)
{
        struct tcp_sock *tp = tcp_sk(sk);
        struct sk_buff *skb;

        printk("tcp_push\n");

        skb = tcp_write_queue_tail(sk);
        if (!skb)
                return;
        if (!(flags & MSG_MORE) || forced_push(tp))
                tcp_mark_push(tp, skb);

        tcp_mark_urg(tp, flags);

        if (tcp_should_autocork(sk, skb, size_goal)) {

                /* avoid atomic op if TSQ_THROTTLED bit is already set */
                if (!test_bit(TSQ_THROTTLED, &sk->sk_tsq_flags)) {
                        NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPAUTOCORKING);
                        set_bit(TSQ_THROTTLED, &sk->sk_tsq_flags);
                }
```

C | Tab Width: 8 | Ln 703, Col 29 | INS

I. 커널 소스 편집

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_write_xmit)

Open | tcp_output.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
 * Send at most one packet when push_one > 0. Temporarily ignore
 * cwnd limit to force at most one packet out when push_one == 2.

 * Returns true, if no segments are in flight and we have queued segments,
 * but cannot send anything now because of SWS or another problem.
 */
static bool tcp_write_xmit(struct sock *sk, unsigned int mss_now, int nonagle,
                           int push_one, gfp_t gfp)
{
        struct tcp_sock *tp = tcp_sk(sk);
        struct sk_buff *skb;
        unsigned int tso_segs, sent_pkts;
        int cwnd_quota;
        int result;
        bool is_cwnd_limited = false, is_rwnd_limited = false;
        u32 max_segs;

        sent_pkts = 0;

        printk("tcp_write_xmit\n");

        tcp_mstamp_refresh(tp);
        if (!push_one) {
                /* Do MTU probing. */
                result = tcp_mtu_probe(sk);
                if (!result) {
                        return false;
                } else if (result > 0) {
                        sent_pkts = 1;
```

C | Tab Width: 8 | Ln 2309, Col 58 | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(tcp_transmit_skb)

Open | tcp_output.c /usr/src/linux-4.15.8/net/ipv4 | Save

```
 * job to build the TCP header, and pass the packet down to
 * IP so it can do the same plus pass the packet off to the
 * device.
 *
 * We are working here with either a clone of the original
 * SKB, or a fresh unique copy made by the retransmit engine.
 */
static int tcp_transmit_skb(struct sock *sk, struct sk_buff *skb, int clone_it,
                            gfp_t gfp_mask)
{
        const struct inet_connection_sock *icsk = inet_csk(sk);
        struct inet_sock *inet;
        struct tcp_sock *tp;
        struct tcp_skb_cb *tcb;
        struct tcp_out_options opts;
        unsigned int tcp_options_size, tcp_header_size;
        struct sk_buff *oskb = NULL;
        struct tcp_md5sig_key *md5;
        struct tcphdr *th;
        int err;

        printk("tcp_transmit_skb\n");

        BUG_ON(!skb || !tcp_skb_pcount(skb));
        tp = tcp_sk(sk);

        if (clone_it) {
                TCP_SKB_CB(skb)->tx.in_flight = TCP_SKB_CB(skb)->end_seq
                        - tp->snd_una;
                oskb = skb;
```

C | Tab Width: 8 | Ln 1041, Col 37 | INS

네트워크 커널 소스(layer 4) 편집

❑ 아래 그림과 같이 printk 문 추가(ip_queue_xmit)

Open | ip_output.c /usr/src/linux-4.15.8/net/ipv4 | Save

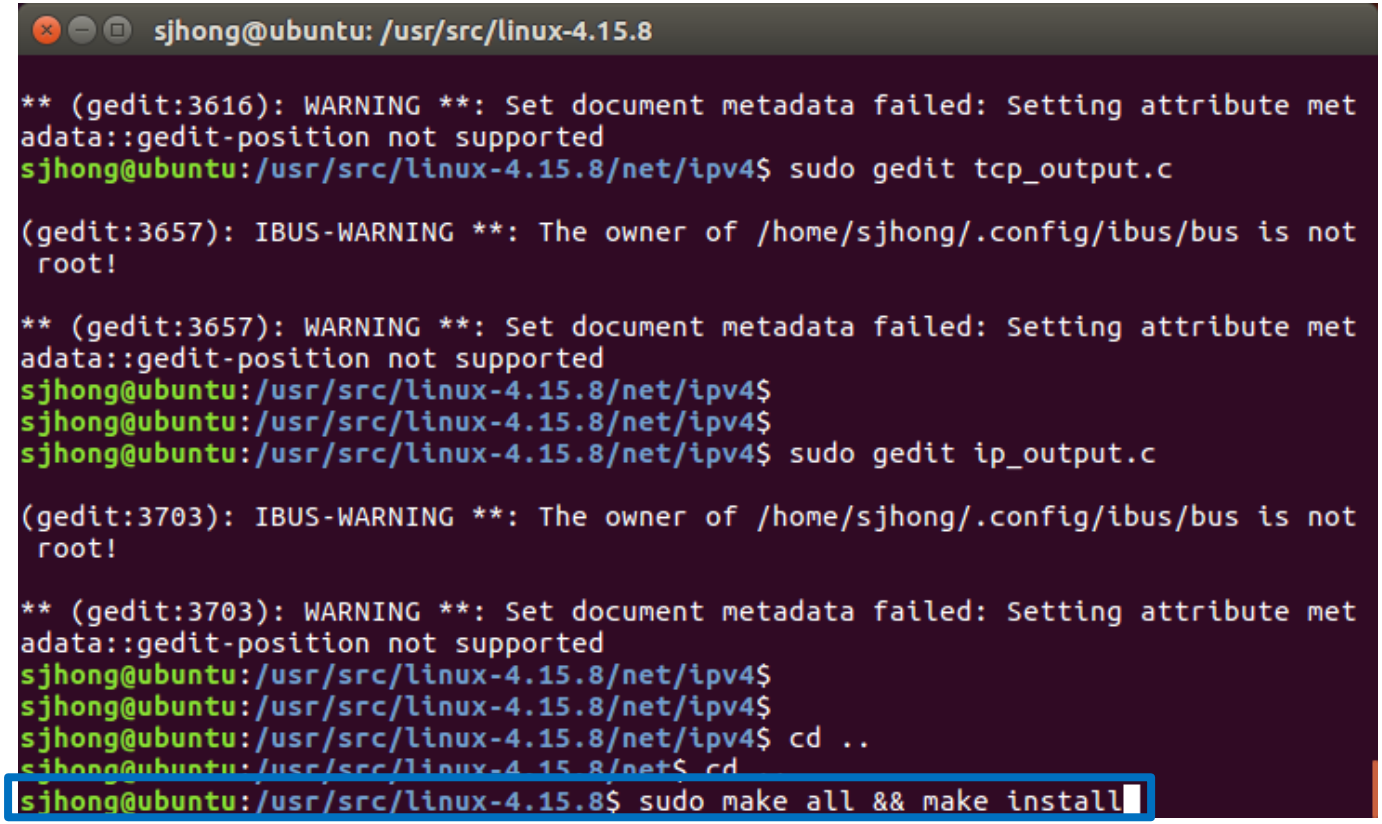
```
                BUILD_BUG_ON(offsetof(typeof(*fl4), daddr) !=
                             offsetof(typeof(*fl4), saddr) + sizeof(fl4->saddr));
        memcpy(&iph->saddr, &fl4->saddr,
               sizeof(fl4->saddr) + sizeof(fl4->daddr));
}

/* Note: skb->sk can be different from sk, in case of tunnels */
int ip_queue_xmit(struct sock *sk, struct sk_buff *skb, struct flowi *fl)
{
        struct inet_sock *inet = inet_sk(sk);
        struct net *net = sock_net(sk);
        struct ip_options_rcu *inet_opt;
        struct flowi4 *fl4;
        struct rtable *rt;
        struct iphdr *iph;
        int res;

        printk("ip_queue_xmit\n");

        /* Skip all of this if the packet is already routed,
         * f.e. by something like SCTP.
         */
        rcu_read_lock();
        inet_opt = rcu_dereference(inet->inet_opt);
        fl4 = &fl->u.ip4;
        rt = skb_rtable(skb);
        if (rt)
                goto packet_routed;

        /* Make sure we can route this packet. */
```

C | Tab Width: 8 | Ln 440, Col 27 | INS

2. 커널 소스 컴파일 및 설치

네트워크 커널 소스 컴파일 및 설치

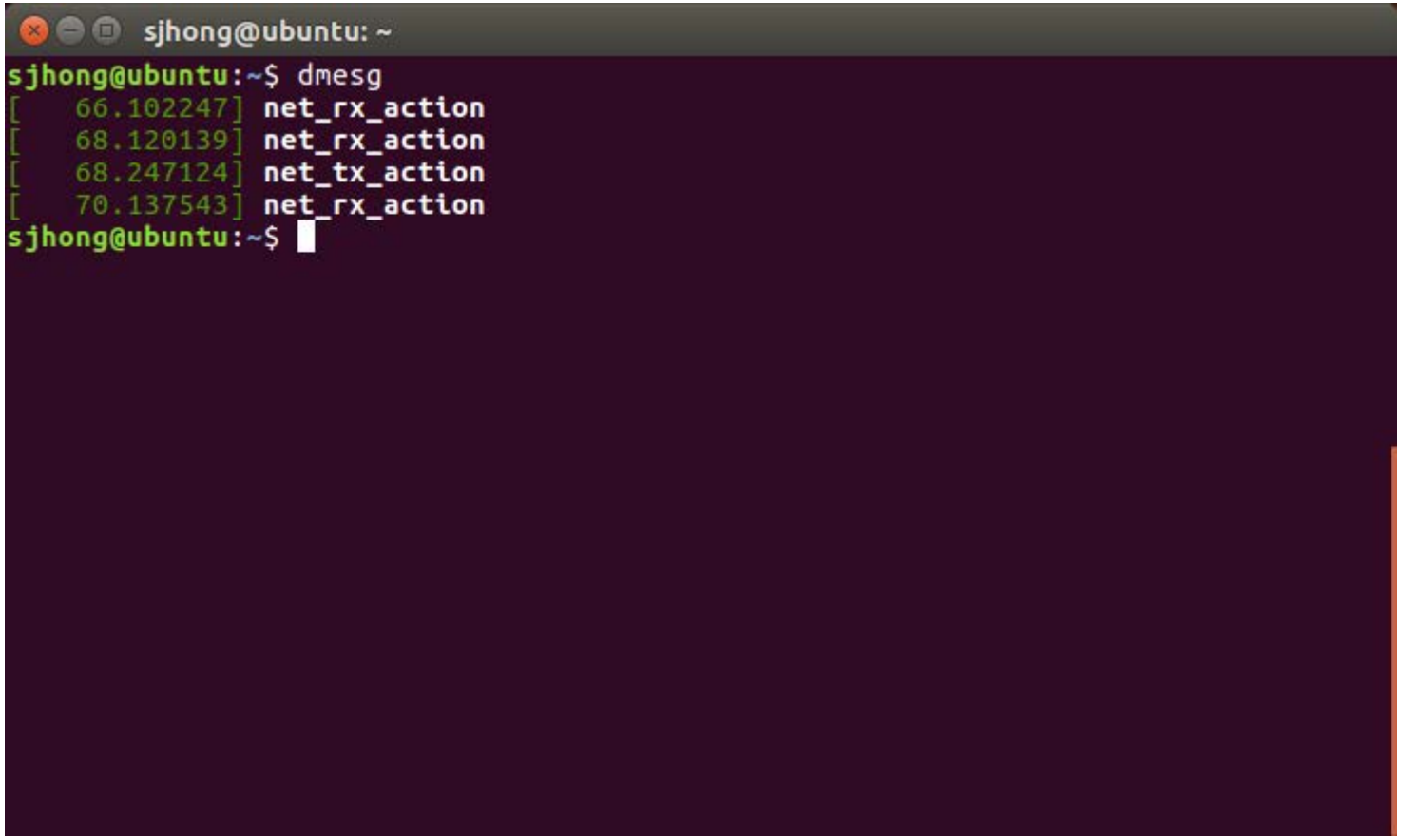
- 아래 그림과 같이 리눅스 커널의 최상위 폴더로 이동후 컴파일 실행
- sudo make all && make install로 컴파일 및 설치



3. 커널 메시지 출력 확인

커널 메시지 출력 확인

- 아래 그림과 같이 dmesg 입력 후 net_rx_action/net_tx_action 메시지가 올라오는 지 확인




Thank you for your attention !!