A lightweight zerocopy notification mechanism

Zijian Zhang, Xiaochun Lu

System Technologies & Engineering, ByteDance

Netdev 0x18@Santa Clara

ByteDance字节跳动

Agenda

- Background
- Problem Statement
- Design of Solution
- Implementation
- Evaluation

Background - history of TCP zerocopy in Linux: TX

Linux had sendfile() support since early days.

This is using ops->sendpage() or ops->sendpage_locked() and available to splice() users, like sendfile() and vmsplice()

In linux-4.14, Willem de Bruijn added MSG_ZEROCOPY support to sendmsg() system call, along with completions sent to the socket error queue.

sendmsg(MSG_ZEROCOPY) is slightly more efficient since it does not have to lock the socket for every page, unlike tcp_sendpage().

Reference: https://netdevconf.info//0x14/pub/slides/62

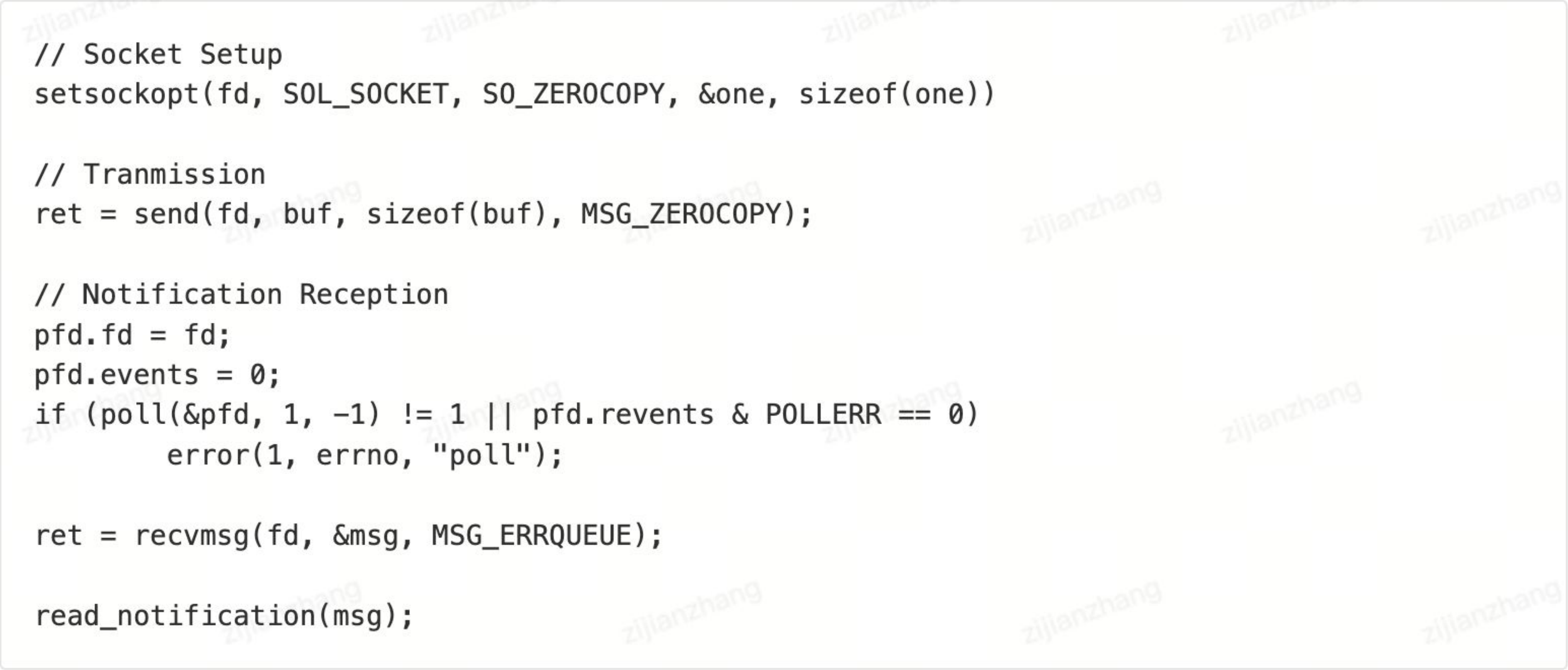
Background - history of TCP zerocopy in Linux: TX

The sendmsg zerocopy needs hardware to support standard SG support and TX checksum offloading.There is no requirements on how memory blocks need to be sized/aligned.

Almost all modern NIC support these (and more)

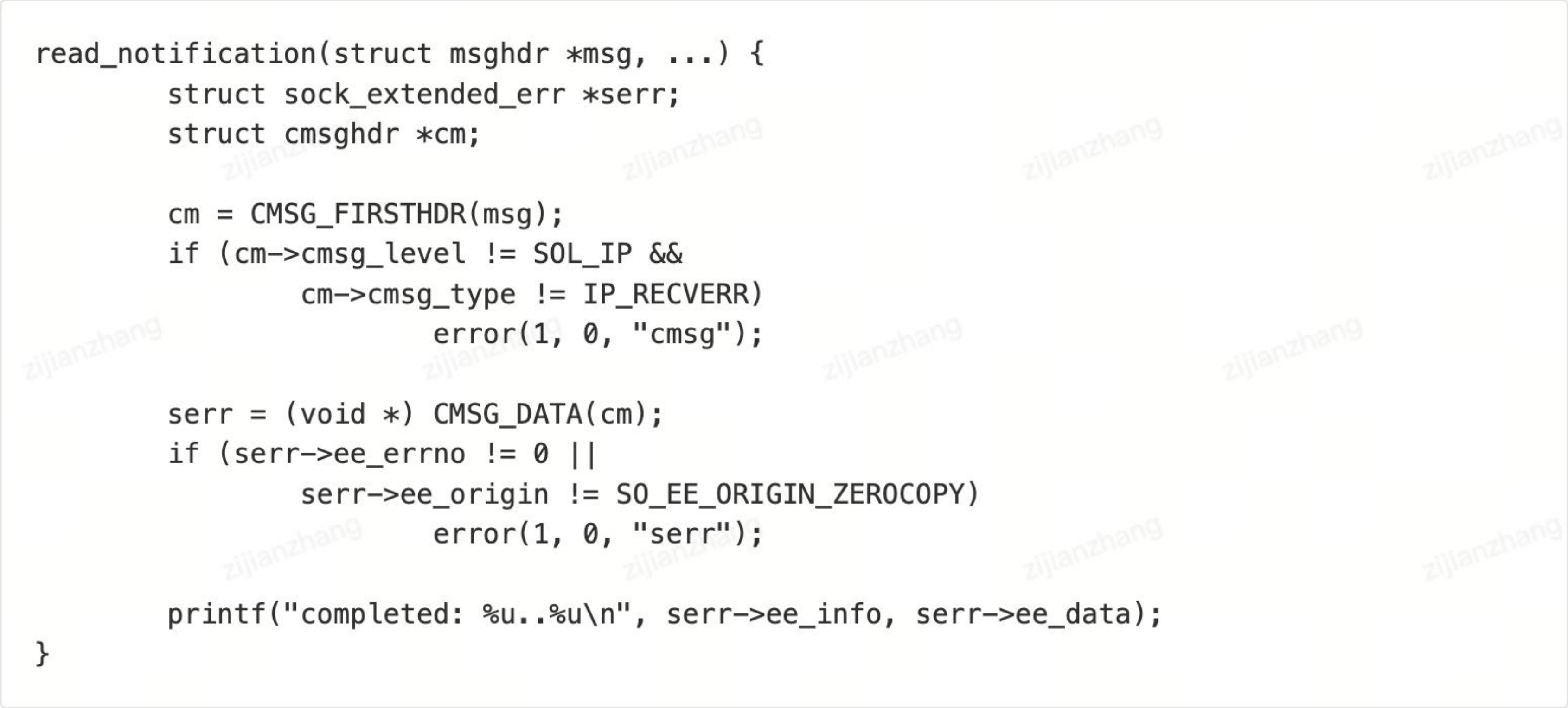
Reference: https://netdevconf.info//0x14/pub/slides/62

Background - MSG_ZEROCOPY notification



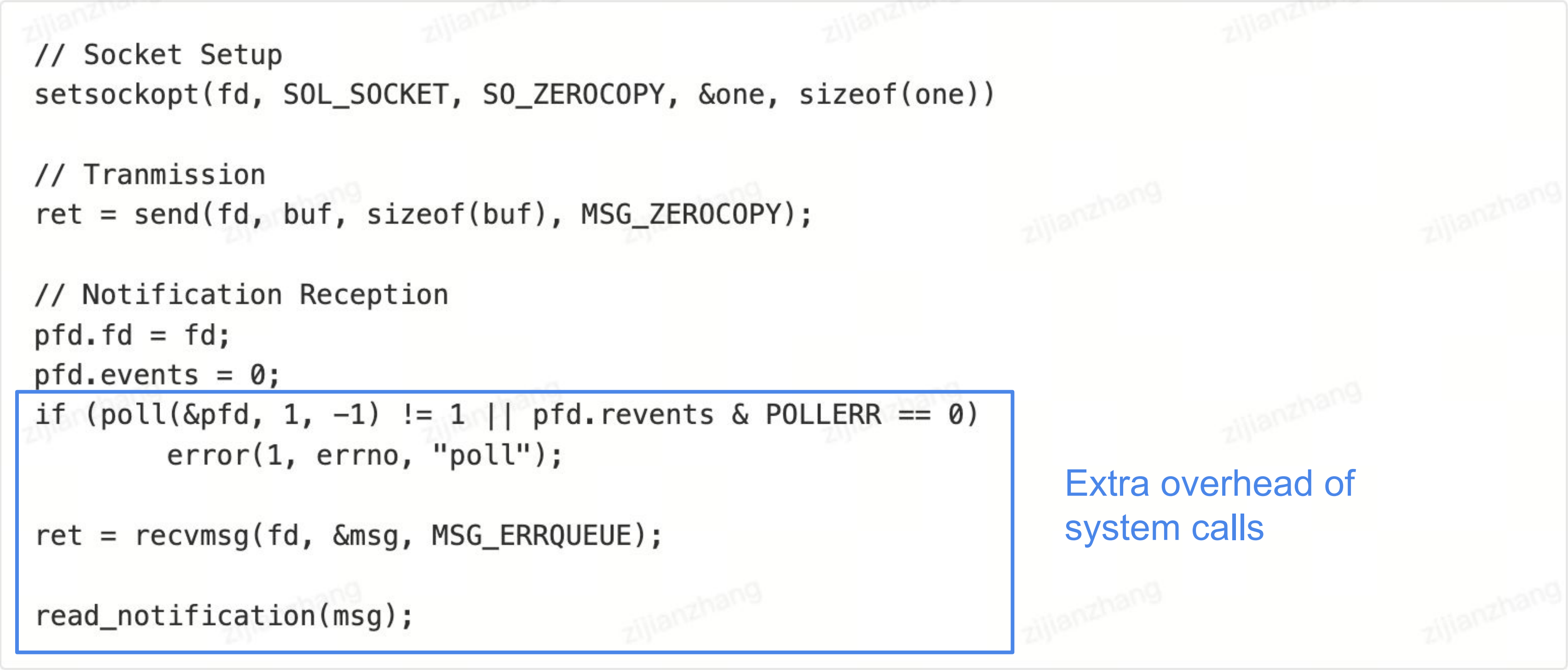
```
// Socket Setup
setsockopt(fd, SOL_SOCKET, SO_ZEROCOPY, &one, sizeof(one))

// Tranmission
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);

// Notification Reception
pfd.fd = fd;
pfd.events = 0;
if (poll(&pfd, 1, -1) != 1 || pfd.revents & POLLERR == 0)
        error(1, errno, "poll");

ret = recvmsg(fd, &msg, MSG_ERRQUEUE);

read_notification(msg);
```

Background - MSG_ZEROCOPY notification



```
read_notification(struct msghdr *msg, ...) {
        struct sock_extended_err *serr;
        struct cmsghdr *cm;

        cm = CMSG_FIRSTHDR(msg);
        if (cm->cmsg_level != SOL_IP &&
                cm->cmsg_type != IP_RECVERR)
                        error(1, 0, "cmsg");

        serr = (void *) CMSG_DATA(cm);
        if (serr->ee_errno != 0 ||
                serr->ee_origin != SO_EE_ORIGIN_ZEROCOPY)
                        error(1, 0, "serr");

        printf("completed: %u..%u\n", serr->ee_info, serr->ee_data);
}
```

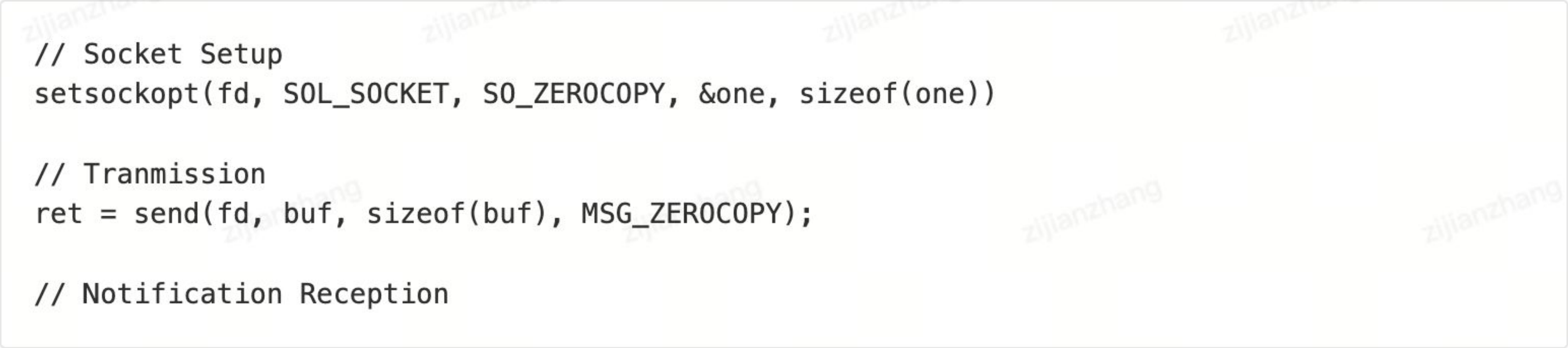
Problem Statement



```
// Socket Setup
setsockopt(fd, SOL_SOCKET, SO_ZEROCOPY, &one, sizeof(one))

// Tranmission
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);

// Notification Reception
pfd.fd = fd;
pfd.events = 0;
if (poll(&pfd, 1, -1) != 1 || pfd.revents & POLLERR == 0)
        error(1, errno, "poll");

ret = recvmsg(fd, &msg, MSG_ERRQUEUE);

read_notification(msg);
```

Extra overhead of system calls

Problem Statement

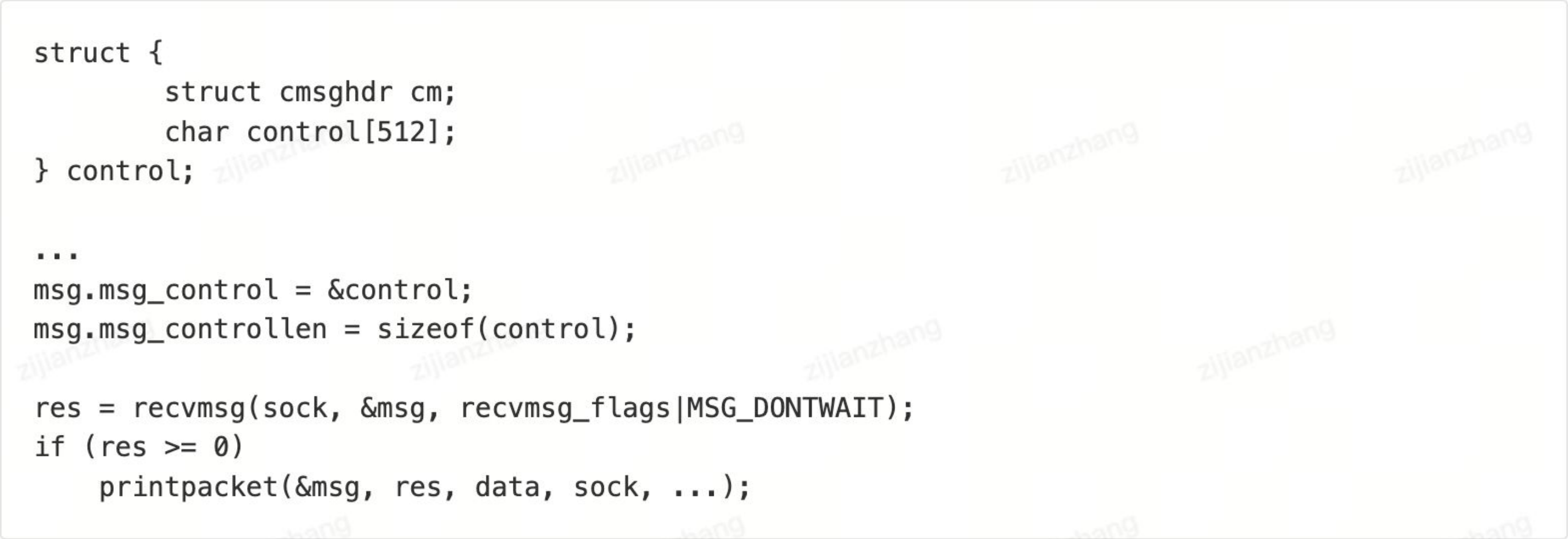


```
// Socket Setup
setsockopt(fd, SOL_SOCKET, SO_ZEROCOPY, &one, sizeof(one))

// Tranmission
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);

// Notification Reception
```

How to return some information back to the user upon returning of sendmsg without introducing extra system calls?

We already have similar mechanism in recvmsg.

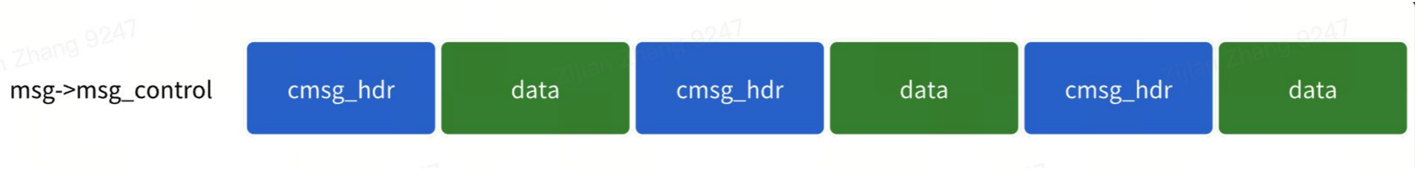
msg_control in recvmsg

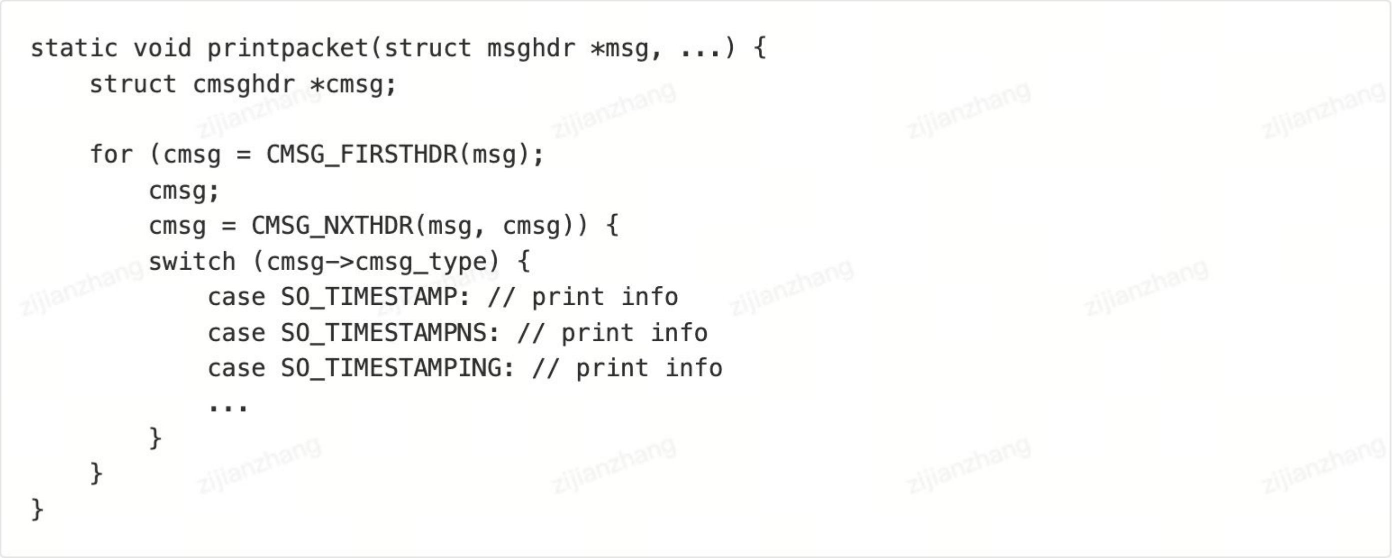


```
struct {
        struct cmsghdr cm;
        char control[512];
} control;

...
msg.msg_control = &control;
msg.msg_controllen = sizeof(control);

res = recvmsg(sock, &msg, recvmsg_flags|MSG_DONTWAIT);
if (res >= 0)
    printpacket(&msg, res, data, sock, ...);
```

Reference: tools/testing/selftests/net/timestamping.c

msg_control in recvmsg

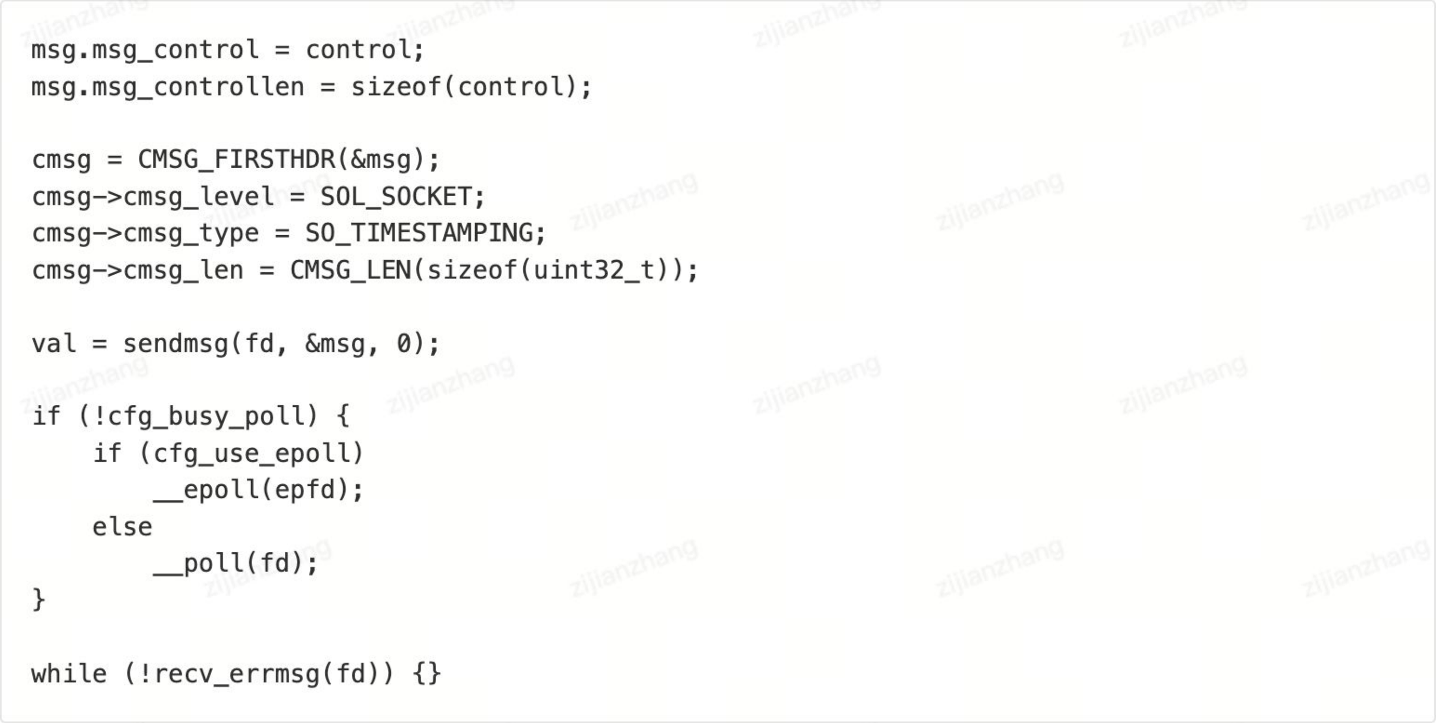




```
static void printpacket(struct msghdr *msg, ...) {
    struct cmsghdr *cmsg;

    for (cmsg = CMSG_FIRSTHDR(msg);
        cmsg;
        cmsg = CMSG_NXTHDR(msg, cmsg)) {
        switch (cmsg->cmsg_type) {
            case SO_TIMESTAMP: // print info
            case SO_TIMESTAMPNS: // print info
            case SO_TIMESTAMPING: // print info
            ...
        }
    }
}
```

Reference: tools/testing/selftests/net/timestamping.c

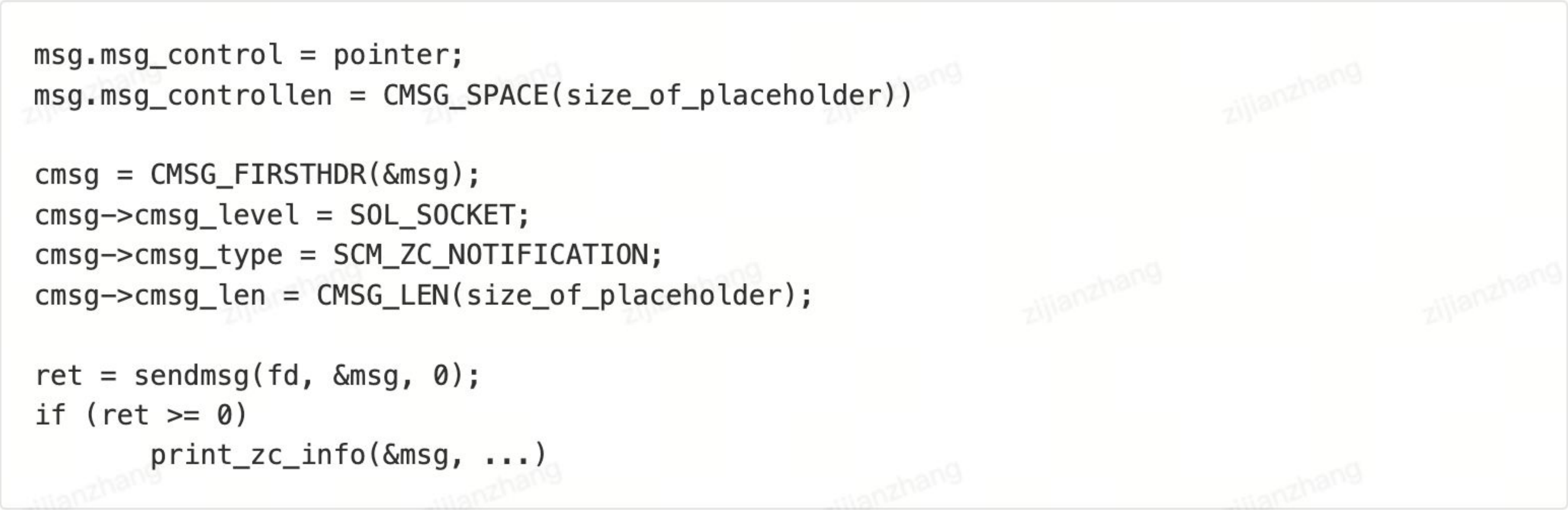
How about msg_control in sendmsg?



```
msg.msg_control = control;
msg.msg_controllen = sizeof(control);

cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SO_TIMESTAMPING;
cmsg->cmsg_len = CMSG_LEN(sizeof(uint32_t));

val = sendmsg(fd, &msg, 0);

if (!cfg_busy_poll) {
    if (cfg_use_epoll)
        __epoll(epfd);
    else
        __poll(fd);
}

while (!recv_errmsg(fd)) {}
```

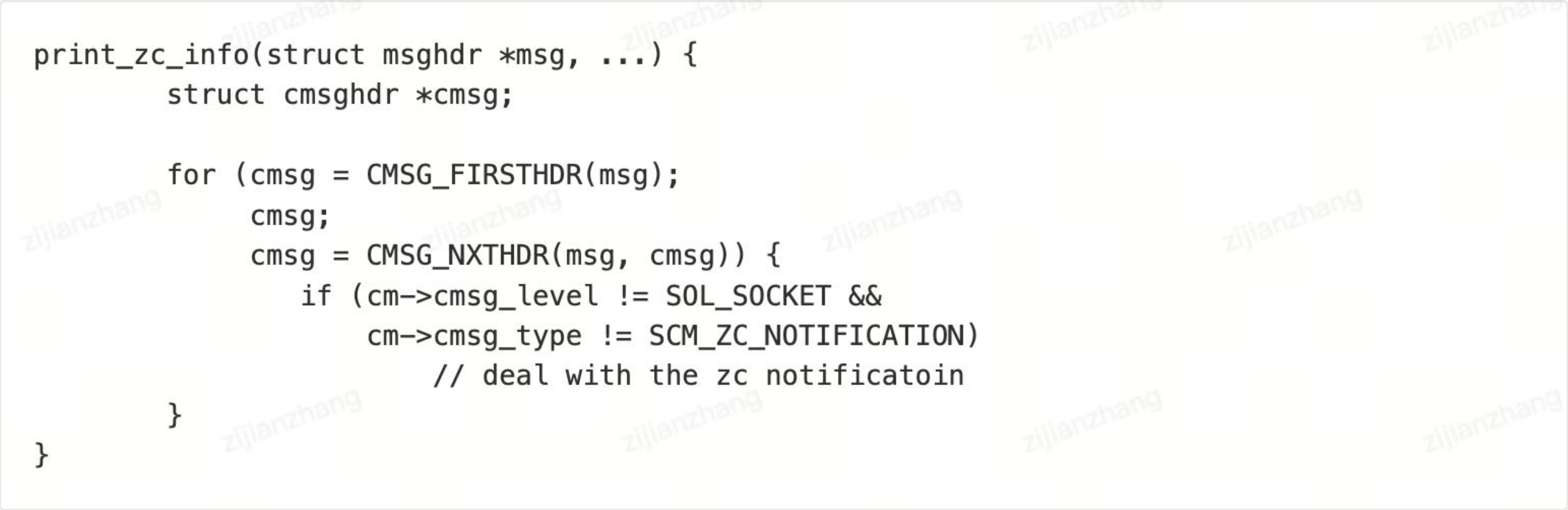
Reference: tools/testing/selftests/net/txtimestamp.c

Design - support msg_control copy back to user in sendmsg



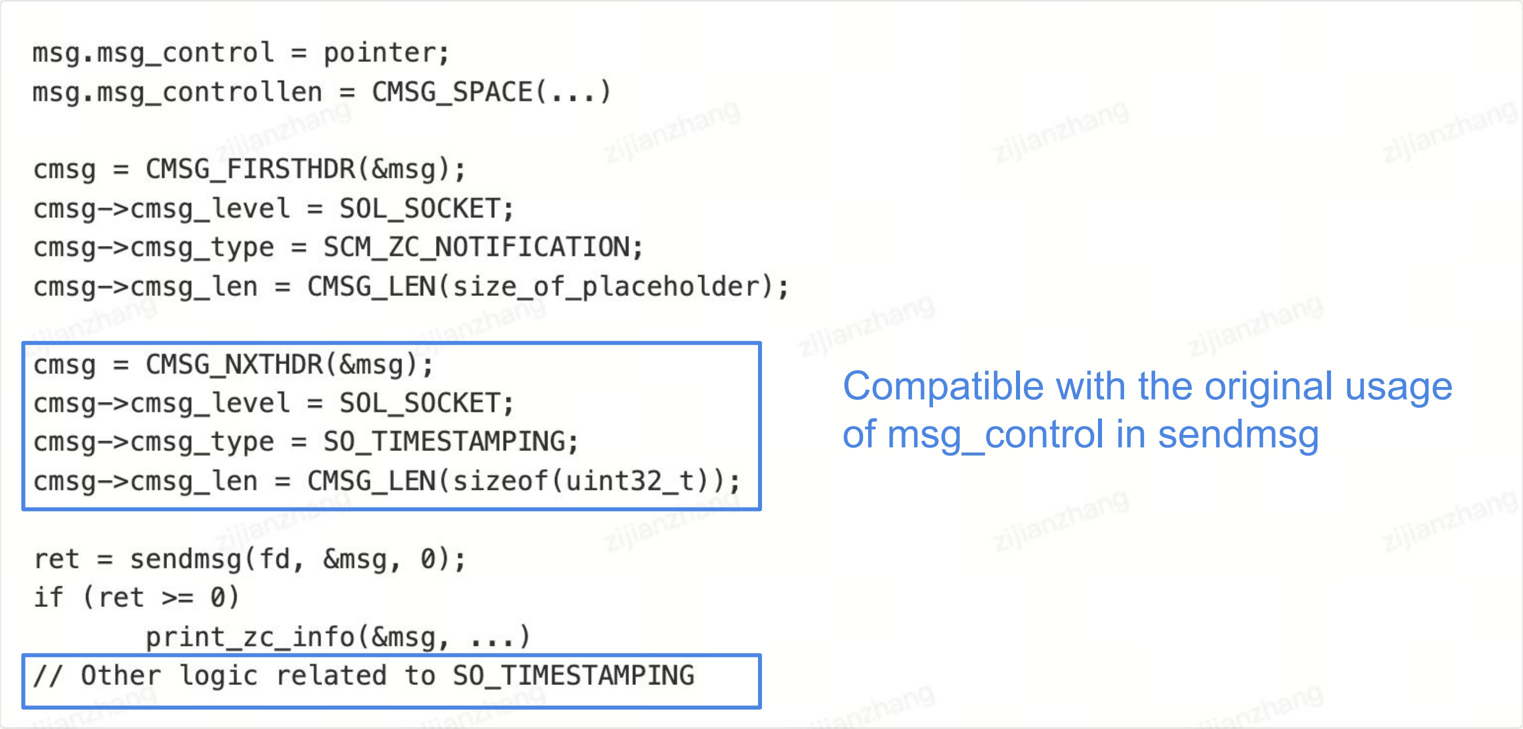
```
msg.msg_control = pointer;
msg.msg_controllen = CMSG_SPACE(size_of_placeholder))

cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_ZC_NOTIFICATION;
cmsg->cmsg_len = CMSG_LEN(size_of_placeholder);

ret = sendmsg(fd, &msg, 0);
if (ret >= 0)
       print_zc_info(&msg, ...)
```

Design - support msg_control copy back to user in sendmsg



```
print_zc_info(struct msghdr *msg, ...) {
        struct cmsghdr *cmsg;

        for (cmsg = CMSG_FIRSTHDR(msg);
             cmsg;
             cmsg = CMSG_NXTHDR(msg, cmsg)) {
                if (cm->cmsg_level != SOL_SOCKET &&
                    cm->cmsg_type != SCM_ZC_NOTIFICATION)
                        // deal with the zc notificatoin
        }
}
```

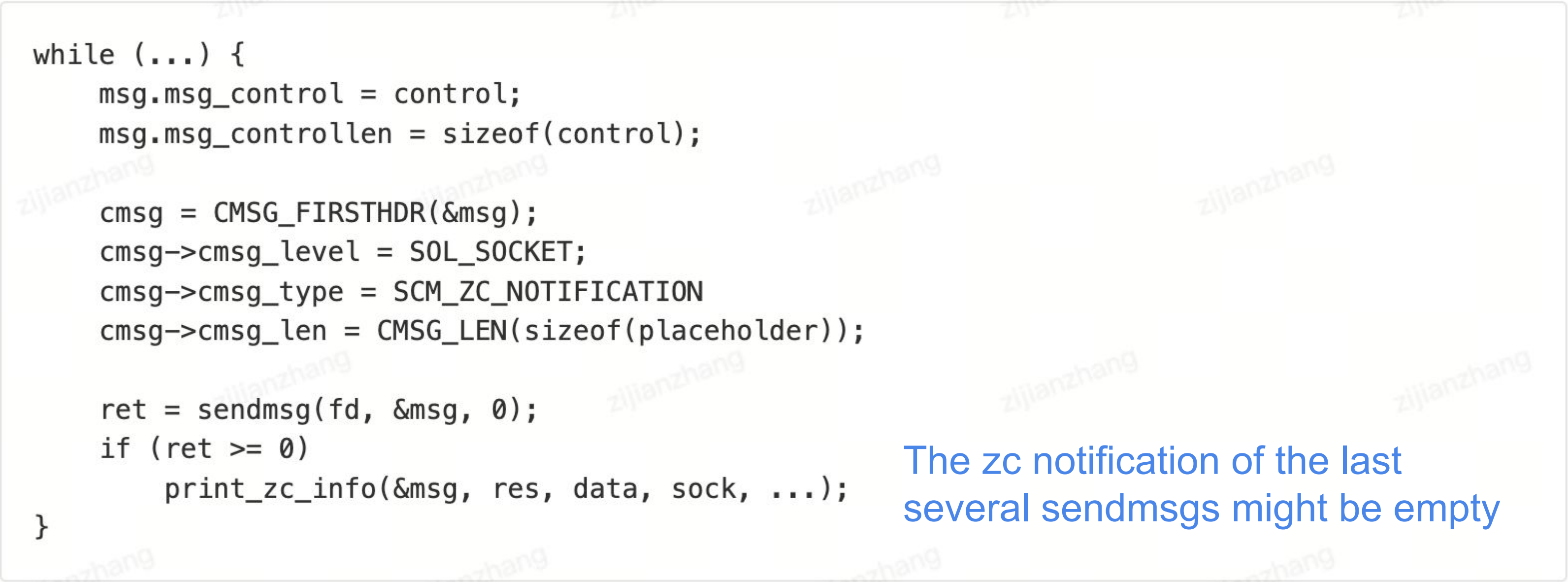
Design - compatibility



```
msg.msg_control = pointer;
msg.msg_controllen = CMSG_SPACE(...)

cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_ZC_NOTIFICATION;
cmsg->cmsg_len = CMSG_LEN(size_of_placeholder);

cmsg = CMSG_NXTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SO_TIMESTAMPING;
cmsg->cmsg_len = CMSG_LEN(sizeof(uint32_t));

ret = sendmsg(fd, &msg, 0);
if (ret >= 0)
       print_zc_info(&msg, ...)
// Other logic related to SO_TIMESTAMPING
```

Compatible with the original usage of msg_control in sendmsg

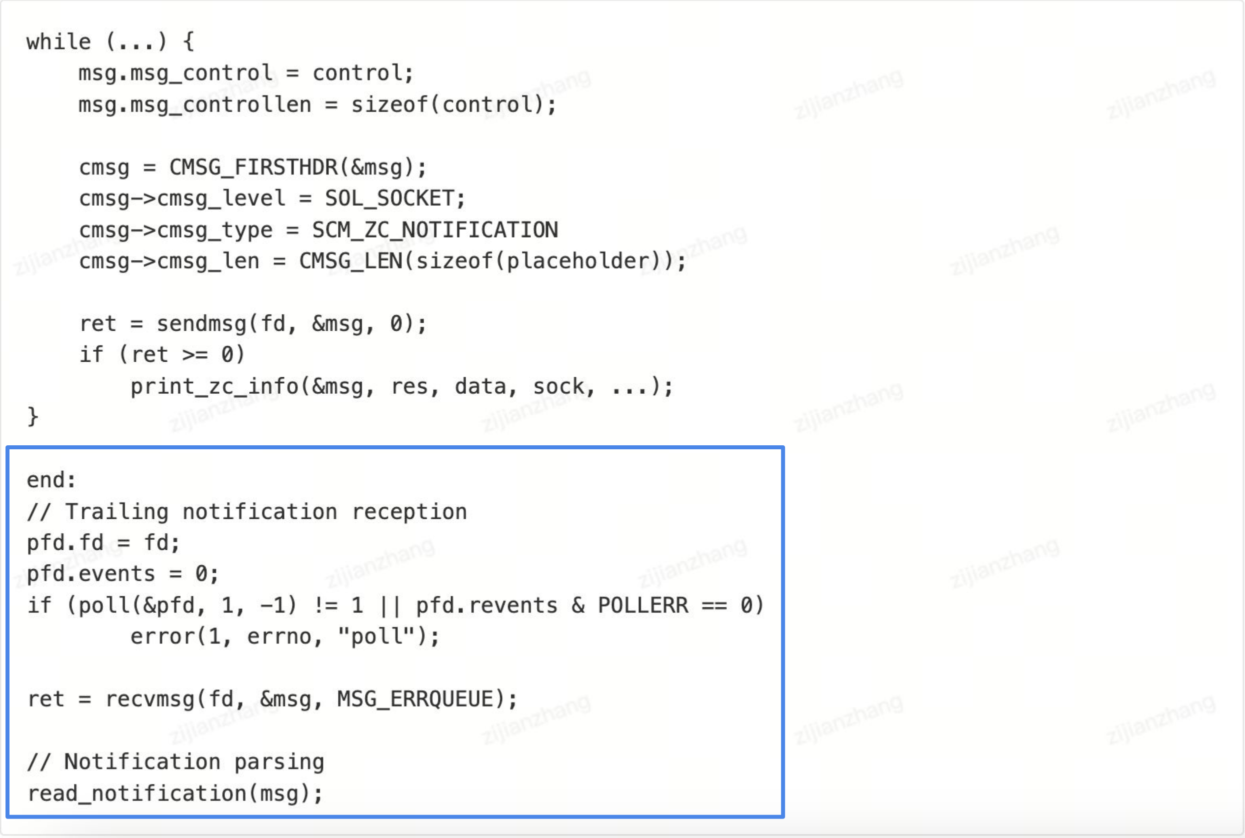
Design - trailing notifications



```
while (...) {
    msg.msg_control = control;
    msg.msg_controllen = sizeof(control);

    cmsg = CMSG_FIRSTHDR(&msg);
    cmsg->cmsg_level = SOL_SOCKET;
    cmsg->cmsg_type = SCM_ZC_NOTIFICATION
    cmsg->cmsg_len = CMSG_LEN(sizeof(placeholder));

    ret = sendmsg(fd, &msg, 0);
    if (ret >= 0)
        print_zc_info(&msg, res, data, sock, ...);
}
```

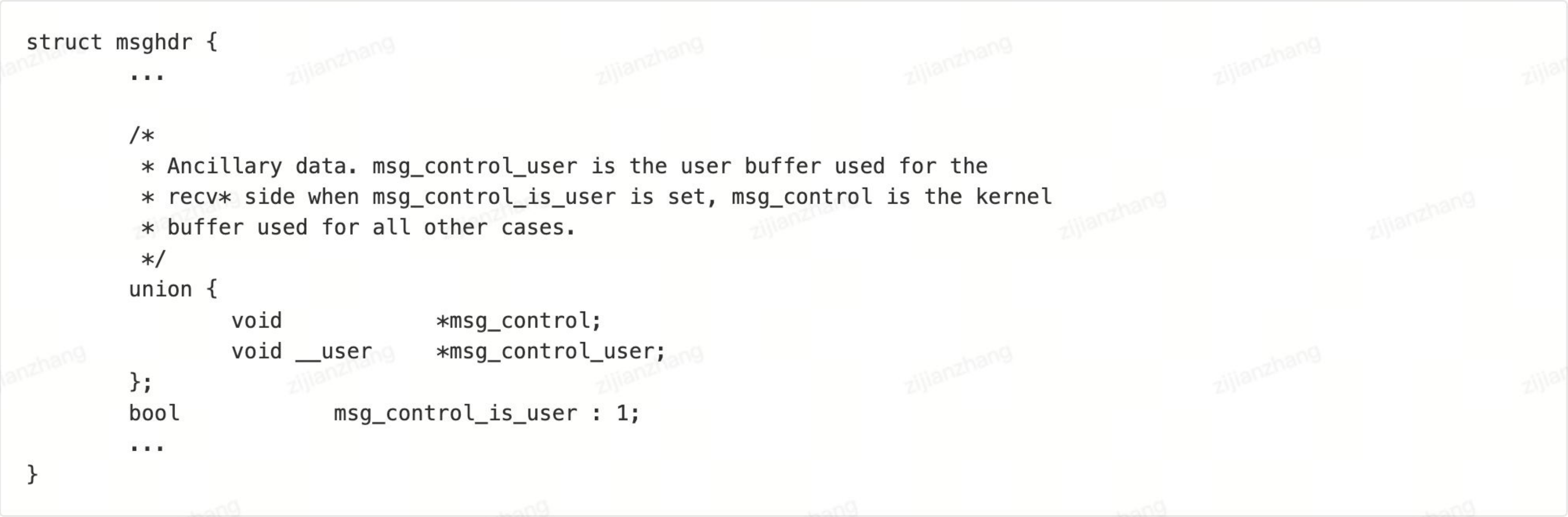
The zc notification of the last several sendmsgs might be empty

Design - user interface - compatible with the original method



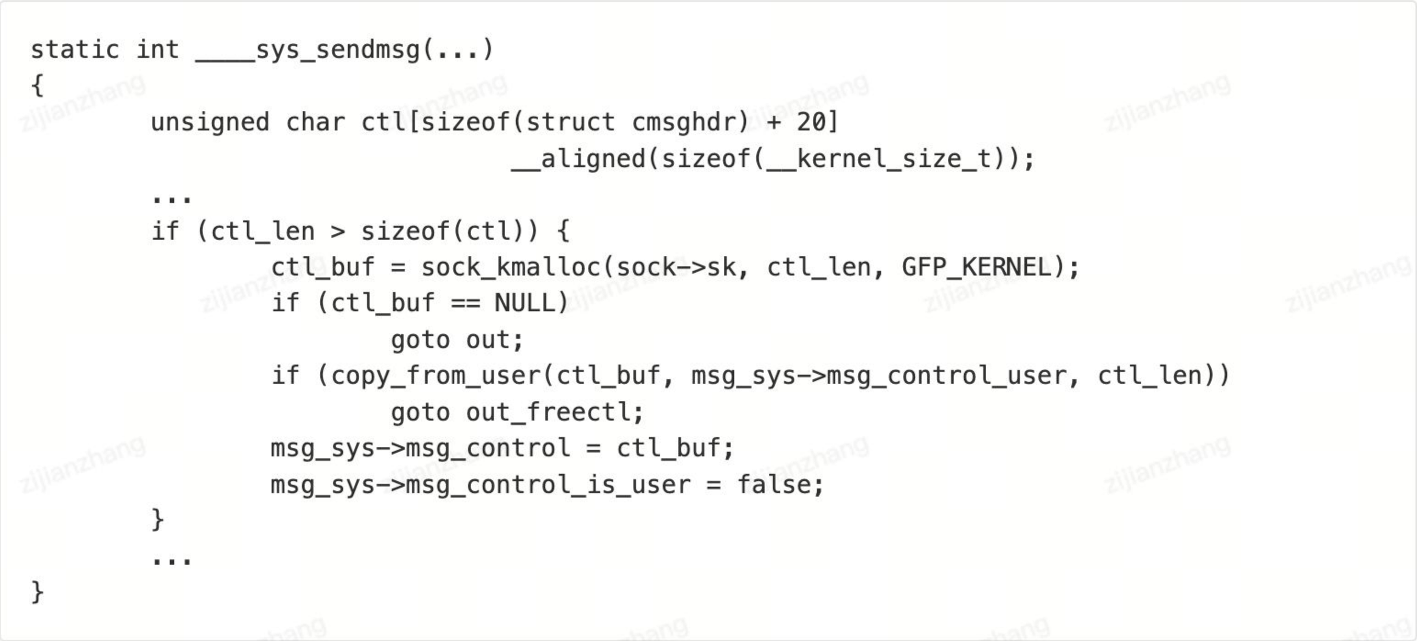
```
while (...) {
    msg.msg_control = control;
    msg.msg_controllen = sizeof(control);

    cmsg = CMSG_FIRSTHDR(&msg);
    cmsg->cmsg_level = SOL_SOCKET;
    cmsg->cmsg_type = SCM_ZC_NOTIFICATION
    cmsg->cmsg_len = CMSG_LEN(sizeof(placeholder));

    ret = sendmsg(fd, &msg, 0);
    if (ret >= 0)
        print_zc_info(&msg, res, data, sock, ...);
}

end:
// Trailing notification reception
pfd.fd = fd;
pfd.events = 0;
if (poll(&pfd, 1, -1) != 1 || pfd.revents & POLLERR == 0)
        error(1, errno, "poll");

ret = recvmsg(fd, &msg, MSG_ERRQUEUE);

// Notification parsing
read_notification(msg);
```

Current msg_control logic in sendmsg



```
struct msghdr {
        ...

        /*
         * Ancillary data. msg_control_user is the user buffer used for the
         * recv* side when msg_control_is_user is set, msg_control is the kernel
         * buffer used for all other cases.
         */
        union {
                void            *msg_control;
                void __user     *msg_control_user;
        };
        bool            msg_control_is_user : 1;
        ...
}
```

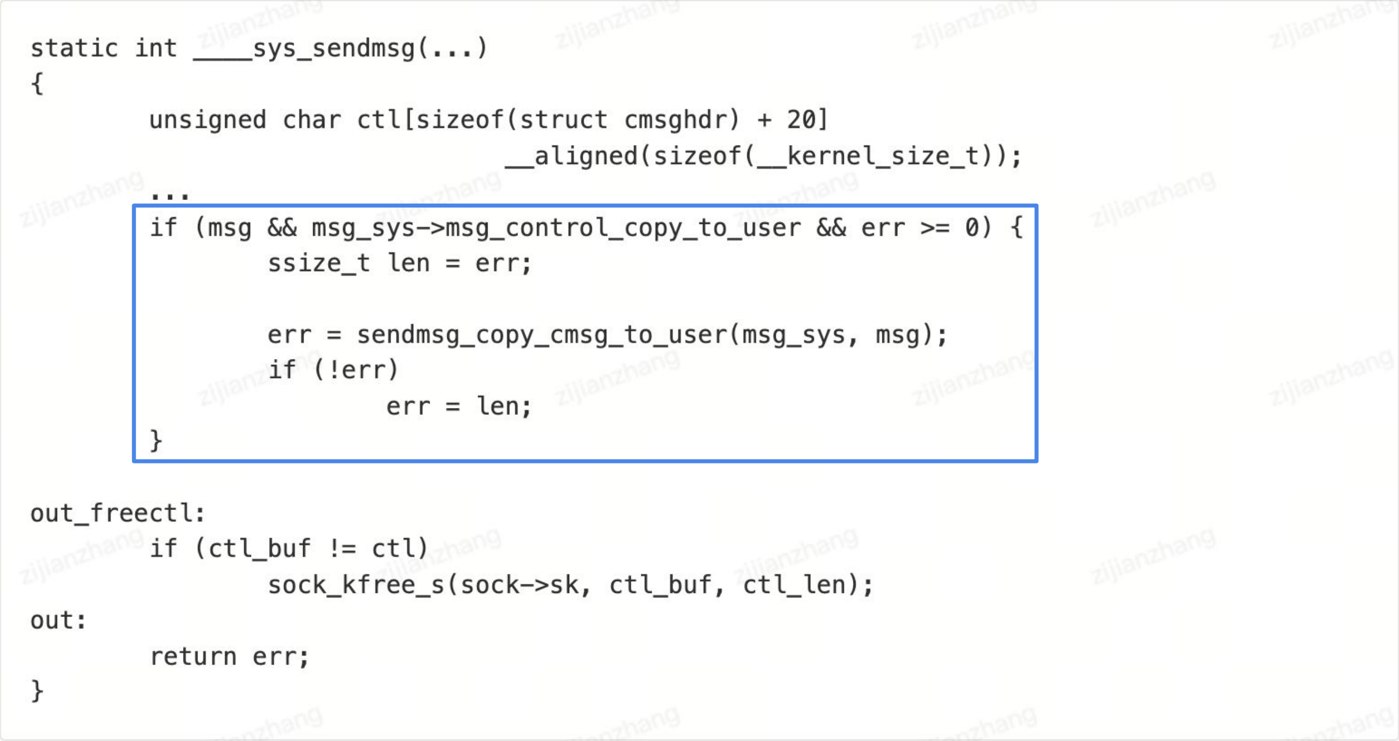
Current msg_control logic in sendmsg



```
static int ____sys_sendmsg(...)
{
        unsigned char ctl[sizeof(struct cmsghdr) + 20]
                                __aligned(sizeof(__kernel_size_t));
        ...
        if (ctl_len > sizeof(ctl)) {
                ctl_buf = sock_kmalloc(sock->sk, ctl_len, GFP_KERNEL);
                if (ctl_buf == NULL)
                        goto out;
                if (copy_from_user(ctl_buf, msg_sys->msg_control_user, ctl_len))
                        goto out_freectl;
                msg_sys->msg_control = ctl_buf;
                msg_sys->msg_control_is_user = false;
        }
        ...
}
```

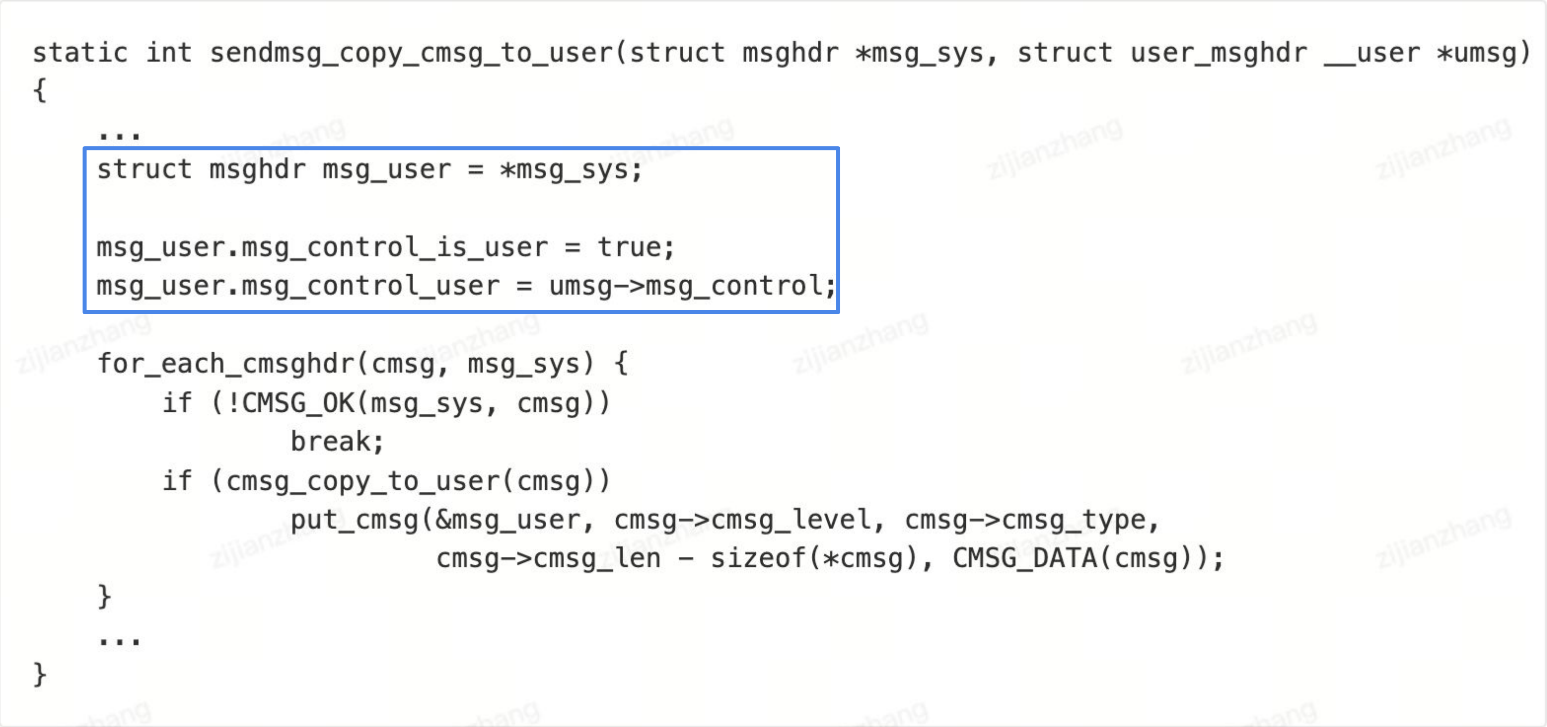
User passed in msg_control_user address is overwritten by a kernel buffer pointer.
For the convenience of further access in the kernel.

Implementation - A generic msg_control copy back framework



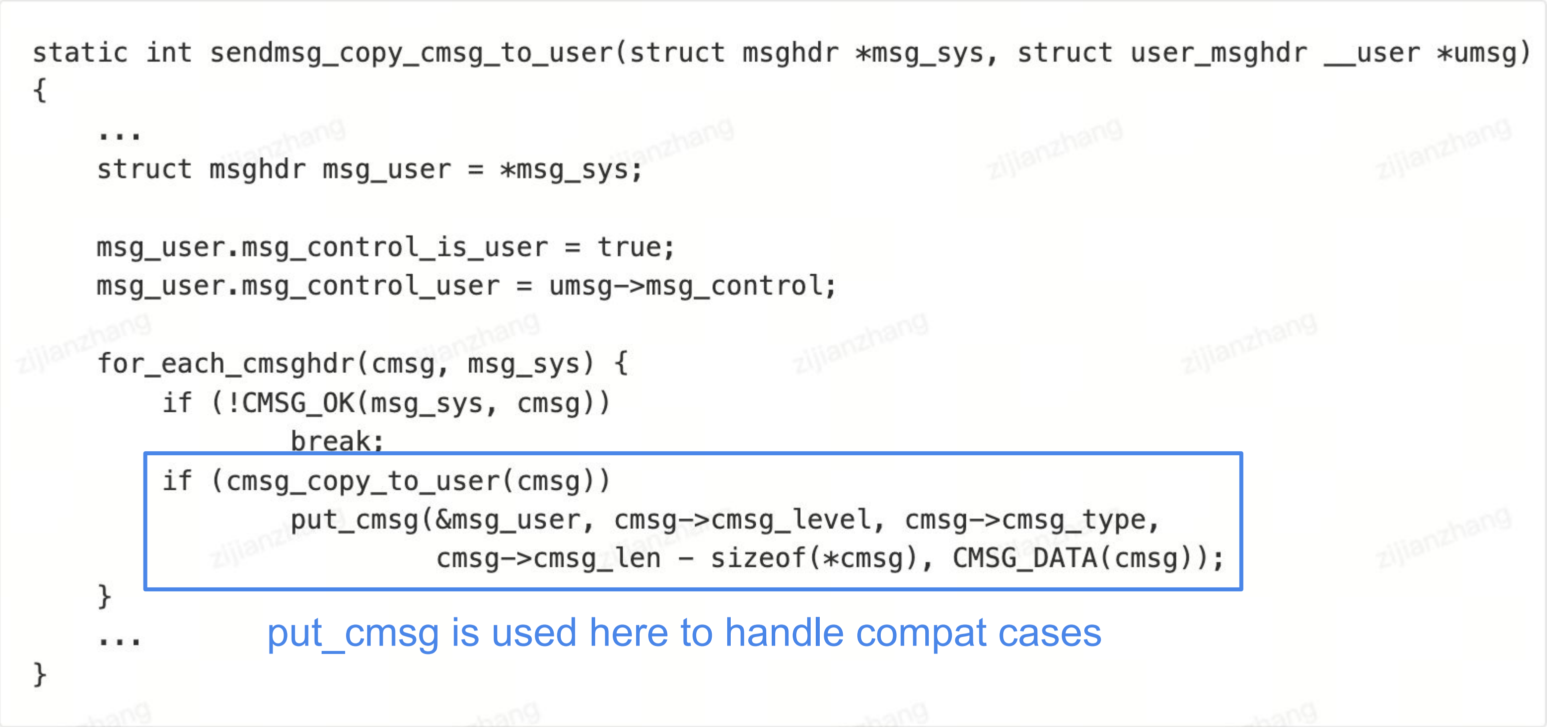
```
static int ____sys_sendmsg(...)
{
        unsigned char ctl[sizeof(struct cmsghdr) + 20]
                                __aligned(sizeof(__kernel_size_t));
        ...
        if (msg && msg_sys->msg_control_copy_to_user && err >= 0) {
                ssize_t len = err;

                err = sendmsg_copy_cmsg_to_user(msg_sys, msg);
                if (!err)
                        err = len;
        }

out_freectl:
        if (ctl_buf != ctl)
                sock_kfree_s(sock->sk, ctl_buf, ctl_len);
out:
        return err;
}
```

Implementation - A generic msg_control copy back framework



```
static int sendmsg_copy_cmsg_to_user(struct msghdr *msg_sys, struct user_msghdr __user *umsg)
{
    ...
    struct msghdr msg_user = *msg_sys;

    msg_user.msg_control_is_user = true;
    msg_user.msg_control_user = umsg->msg_control;

    for_each_cmsghdr(cmsg, msg_sys) {
        if (!CMSG_OK(msg_sys, cmsg))
                break;
        if (cmsg_copy_to_user(cmsg))
                put_cmsg(&msg_user, cmsg->cmsg_level, cmsg->cmsg_type,
                         cmsg->cmsg_len - sizeof(*cmsg), CMSG_DATA(cmsg));
    }
    ...
}
```

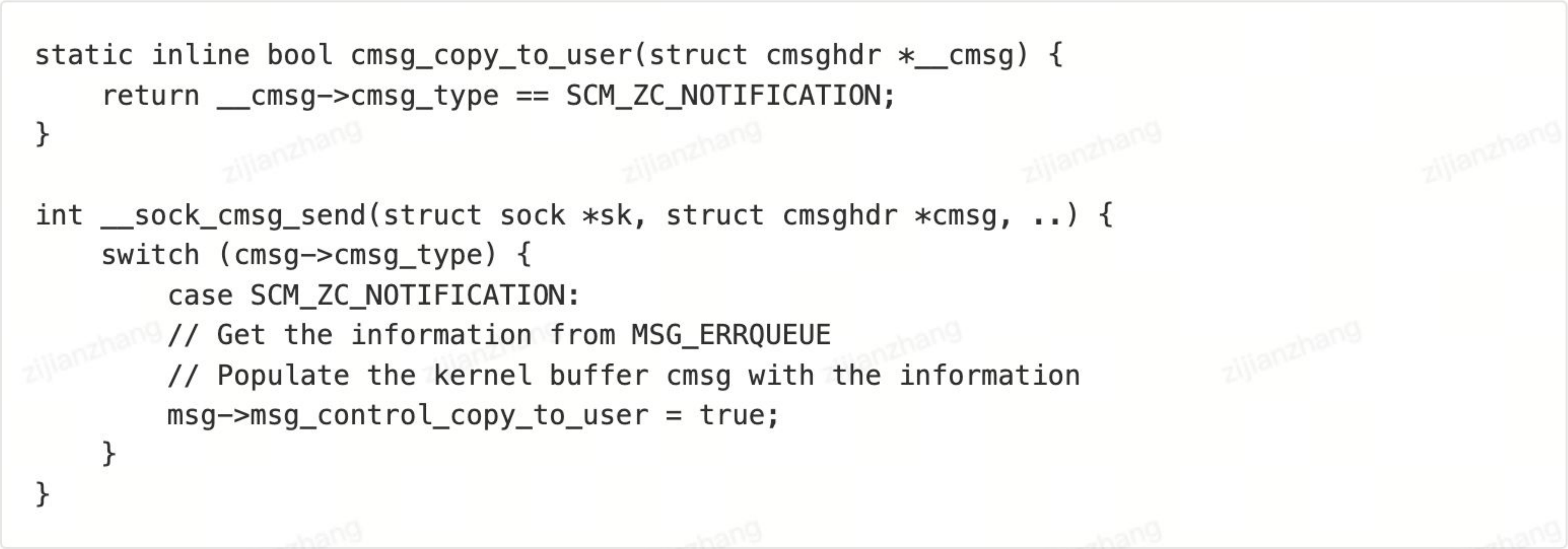
Implementation - A generic msg_control copy back framework



```
static int sendmsg_copy_cmsg_to_user(struct msghdr *msg_sys, struct user_msghdr __user *umsg)
{
    ...
    struct msghdr msg_user = *msg_sys;

    msg_user.msg_control_is_user = true;
    msg_user.msg_control_user = umsg->msg_control;

    for_each_cmsghdr(cmsg, msg_sys) {
        if (!CMSG_OK(msg_sys, cmsg))
                break;
        if (cmsg_copy_to_user(cmsg))
                put_cmsg(&msg_user, cmsg->cmsg_level, cmsg->cmsg_type,
                         cmsg->cmsg_len - sizeof(*cmsg), CMSG_DATA(cmsg));
    }
    ...
}
```

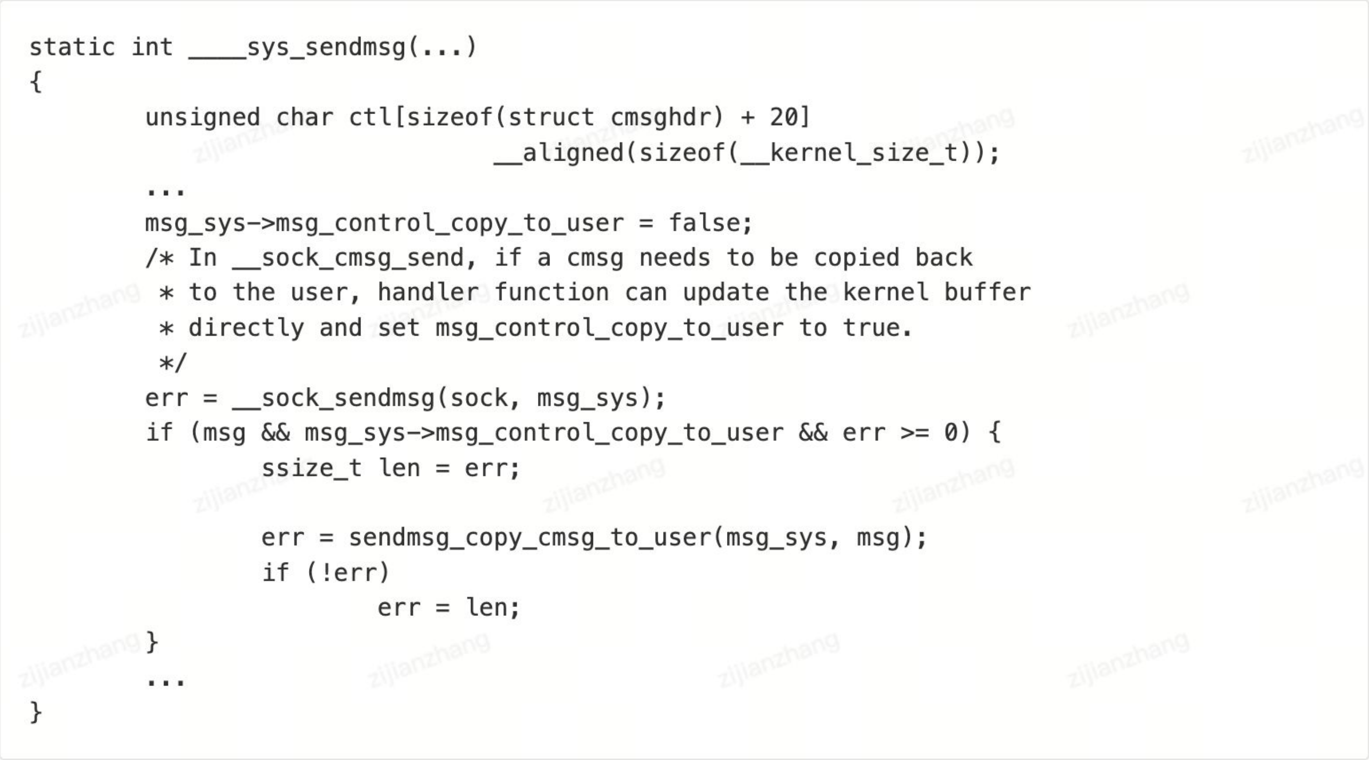
put_cmsg is used here to handle compat cases

Implementation - how to make use of the framework in zerocopy?



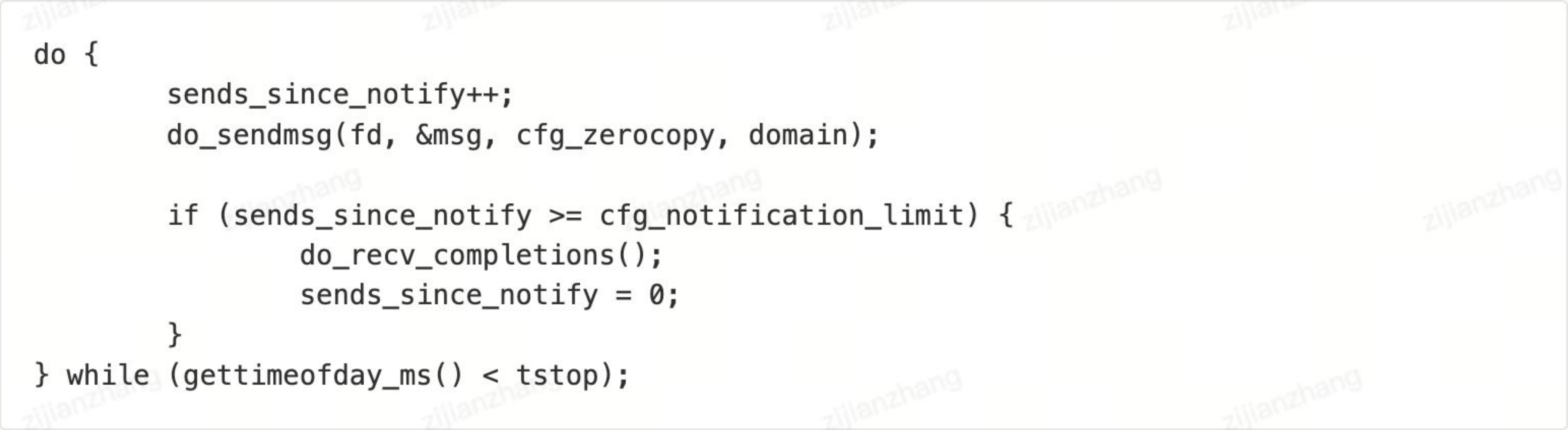
```
static inline bool cmsg_copy_to_user(struct cmsghdr *__cmsg) {
    return __cmsg->cmsg_type == SCM_ZC_NOTIFICATION;
}

int __sock_cmsg_send(struct sock *sk, struct cmsghdr *cmsg, ..) {
    switch (cmsg->cmsg_type) {
        case SCM_ZC_NOTIFICATION:
        // Get the information from MSG_ERRQUEUE
        // Populate the kernel buffer cmsg with the information
        msg->msg_control_copy_to_user = true;
    }
}
```

Implementation - overall



```
static int ____sys_sendmsg(...)
{
        unsigned char ctl[sizeof(struct cmsghdr) + 20]
                                __aligned(sizeof(__kernel_size_t));
        ...
        msg_sys->msg_control_copy_to_user = false;
        /* In __sock_cmsg_send, if a cmsg needs to be copied back
         * to the user, handler function can update the kernel buffer
         * directly and set msg_control_copy_to_user to true.
         */
        err = __sock_sendmsg(sock, msg_sys);
        if (msg && msg_sys->msg_control_copy_to_user && err >= 0) {
                ssize_t len = err;

                err = sendmsg_copy_cmsg_to_user(msg_sys, msg);
                if (!err)
                        err = len;
        }
        ...
}
```

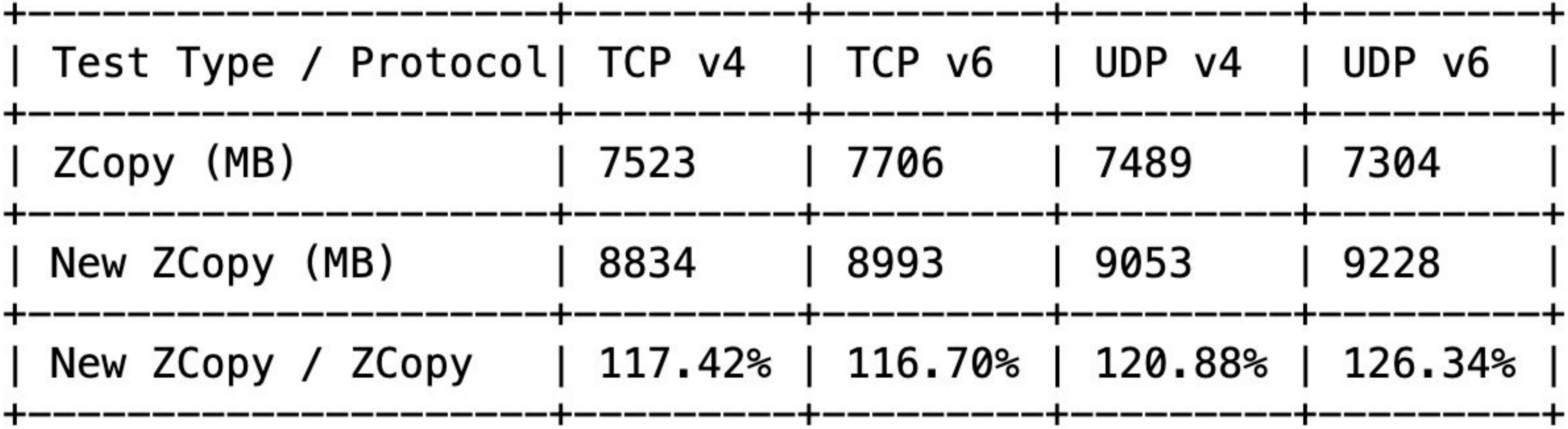
Evaluation - msg_zerocopy selftest



```
do {
        sends_since_notify++;
        do_sendmsg(fd, &msg, cfg_zerocopy, domain);

        if (sends_since_notify >= cfg_notification_limit) {
                do_recv_completions();
                sends_since_notify = 0;
        }
} while (gettimeofday_ms() < tstop);
```

Reference: tools/testing/selftests/net/msg_zerocopy.c

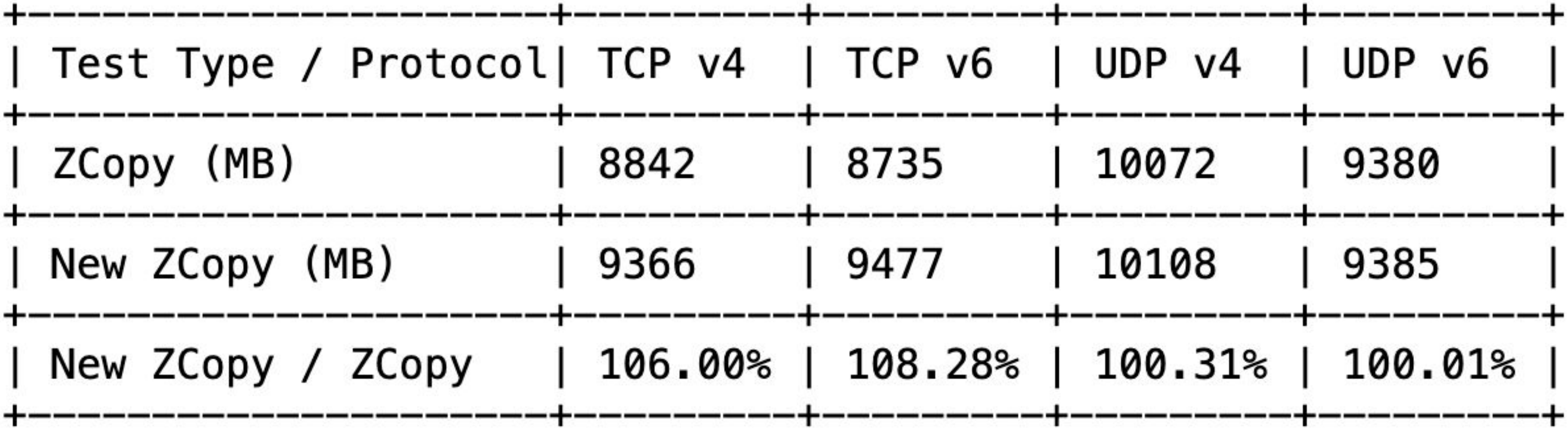
Evaluation - throughput performance - notification interval = 1



```
+---------------------+---------+---------+---------+---------+
| Test Type / Protocol| TCP v4  | TCP v6  | UDP v4  | UDP v6  |
+---------------------+---------+---------+---------+---------+
| ZCopy (MB)          | 7523    | 7706    | 7489    | 7304    |
+---------------------+---------+---------+---------+---------+
| New ZCopy (MB)      | 8834    | 8993    | 9053    | 9228    |
+---------------------+---------+---------+---------+---------+
| New ZCopy / ZCopy   | 117.42% | 116.70% | 120.88% | 126.34% |
+---------------------+---------+---------+---------+---------+
```

Reference: https://lore.kernel.org/all/20240708210405.870930-4-zijianzhang@bytedance.com/

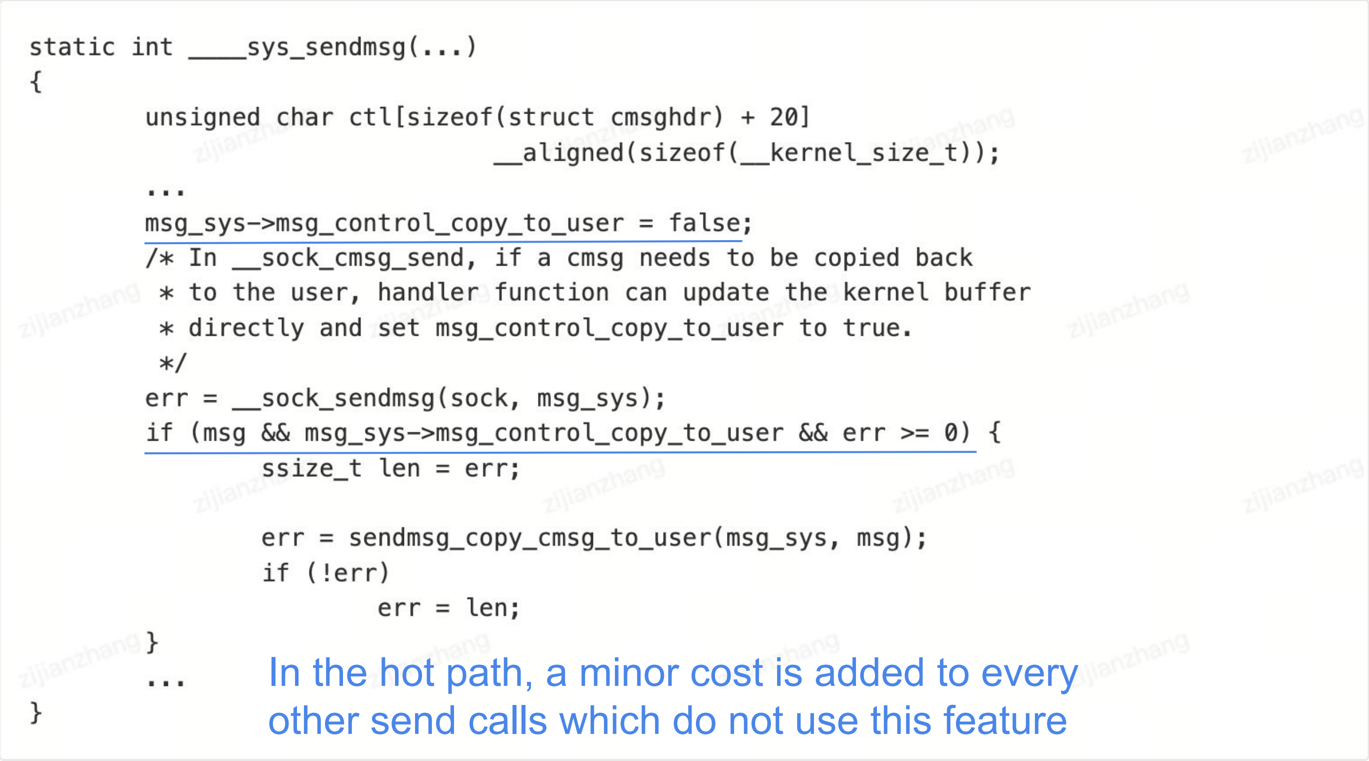
Evaluation - throughput performance - notification interval = 32



```
+-----------------------+----------+----------+----------+----------+
| Test Type / Protocol| TCP v4   | TCP v6   | UDP v4   | UDP v6   |
+-----------------------+----------+----------+----------+----------+
| ZCopy (MB)            | 8842     | 8735     | 10072    | 9380     |
+-----------------------+----------+----------+----------+----------+
| New ZCopy (MB)        | 9366     | 9477     | 10108    | 9385     |
+-----------------------+----------+----------+----------+----------+
| New ZCopy / ZCopy     | 106.00% | 108.28% | 100.31% | 100.01% |
+-----------------------+----------+----------+----------+----------+
```

Reference: https://lore.kernel.org/all/20240708210405.870930-4-zijianzhang@bytedance.com/

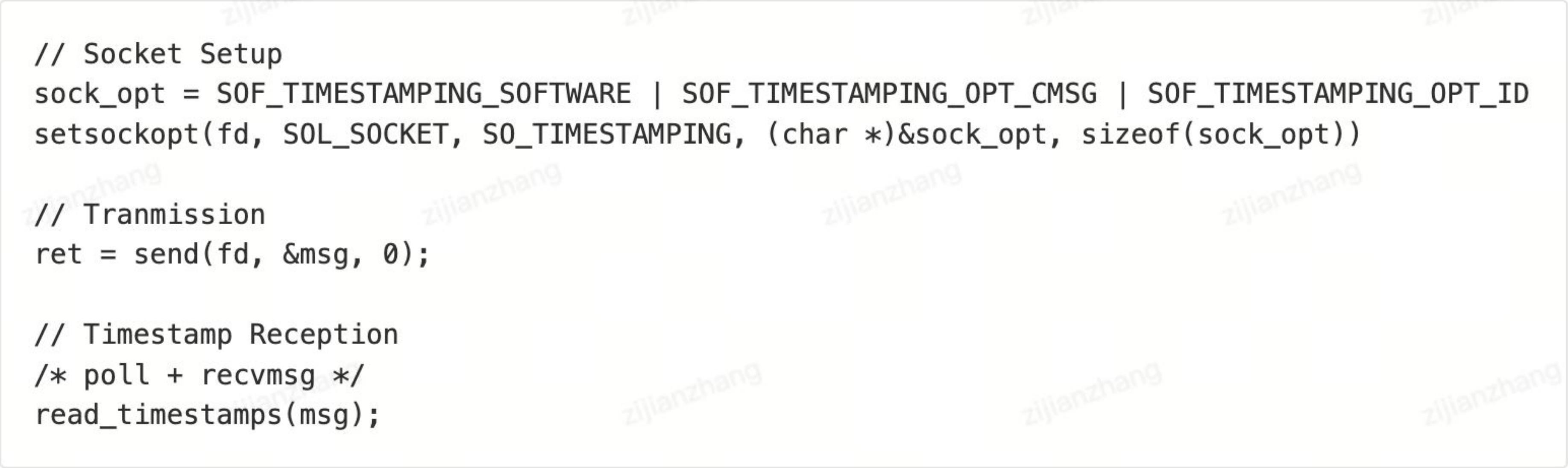
Evaluation - overhead introduced



```
static int ____sys_sendmsg(...)
{
        unsigned char ctl[sizeof(struct cmsghdr) + 20]
                                __aligned(sizeof(__kernel_size_t));
        ...
        msg_sys->msg_control_copy_to_user = false;
        /* In __sock_cmsg_send, if a cmsg needs to be copied back
         * to the user, handler function can update the kernel buffer
         * directly and set msg_control_copy_to_user to true.
         */
        err = __sock_sendmsg(sock, msg_sys);
        if (msg && msg_sys->msg_control_copy_to_user && err >= 0) {
                ssize_t len = err;

                err = sendmsg_copy_cmsg_to_user(msg_sys, msg);
                if (!err)
                        err = len;
        }
        ...
}
```

In the hot path, a minor cost is added to every other send calls which do not use this feature

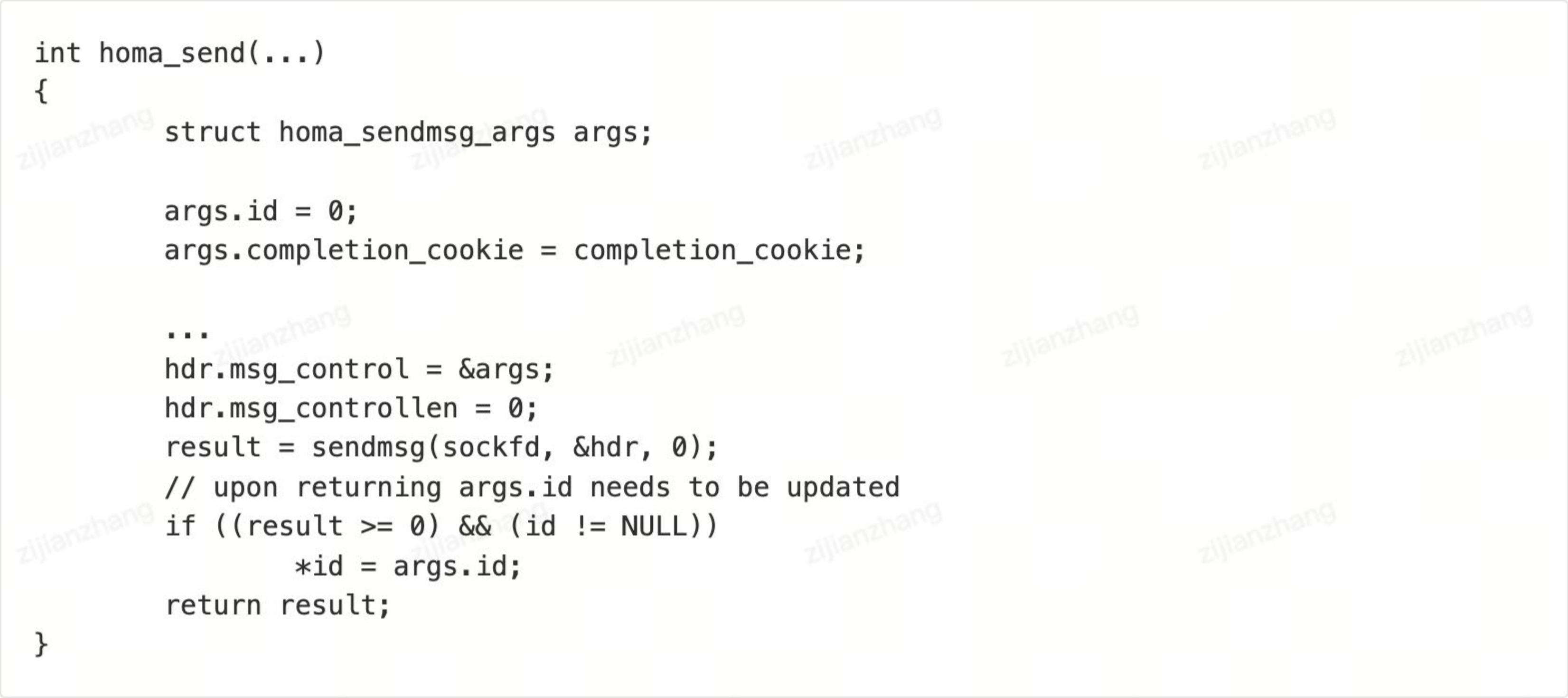
Next step - other possible use cases - timestamp



```
// Socket Setup
sock_opt = SOF_TIMESTAMPING_SOFTWARE | SOF_TIMESTAMPING_OPT_CMSG | SOF_TIMESTAMPING_OPT_ID
setsockopt(fd, SOL_SOCKET, SO_TIMESTAMPING, (char *)&sock_opt, sizeof(sock_opt))

// Tranmission
ret = send(fd, &msg, 0);

// Timestamp Reception
/* poll + recvmsg */
read_timestamps(msg);
```

Reference: tools/testing/selftests/net/txtimestamp.c

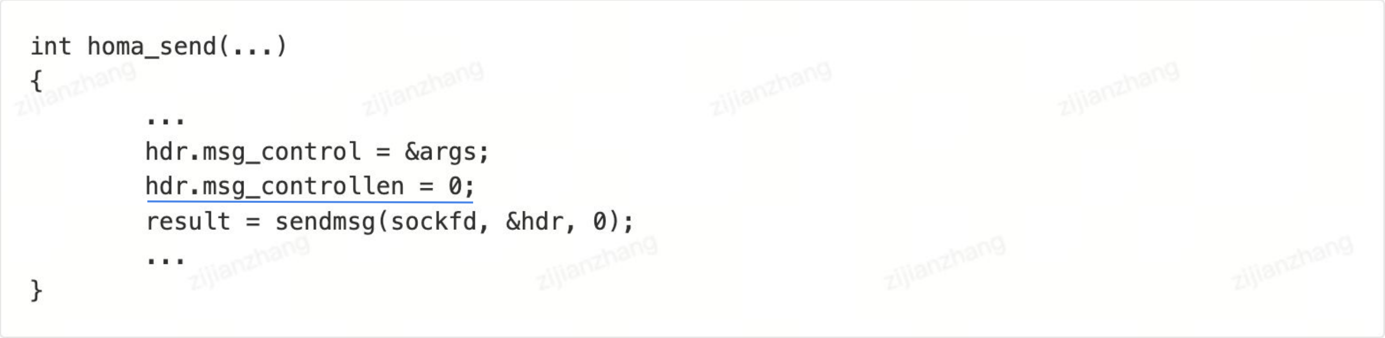
Next step - other possible use cases - Homa



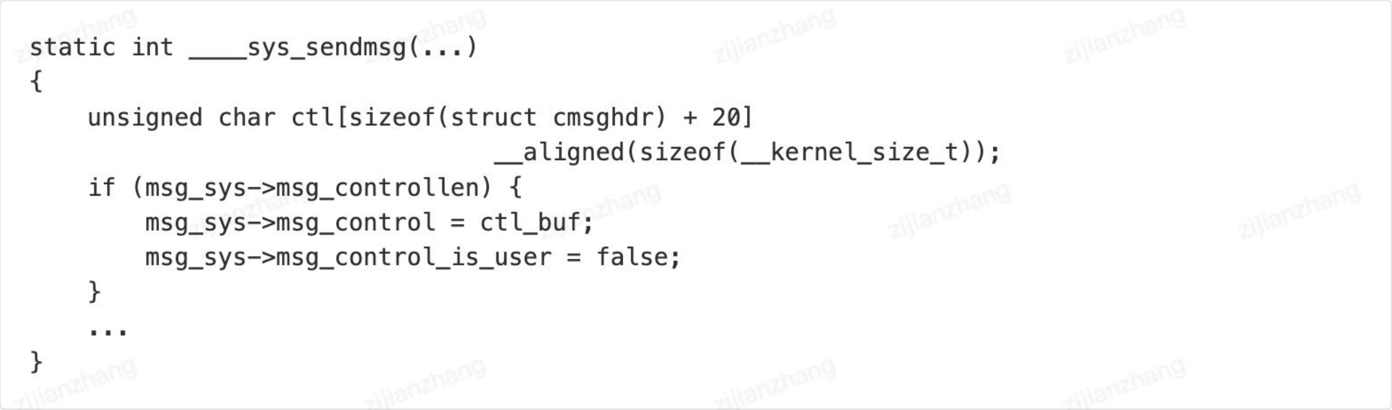
```
int homa_send(...)
{
        struct homa_sendmsg_args args;

        args.id = 0;
        args.completion_cookie = completion_cookie;

        ...
        hdr.msg_control = &args;
        hdr.msg_controllen = 0;
        result = sendmsg(sockfd, &hdr, 0);
        // upon returning args.id needs to be updated
        if ((result >= 0) && (id != NULL))
                *id = args.id;
        return result;
}
```

Reference: https://github.com/PlatformLab/HomaModule/blob/main/homa_api.c

Next step - other possible use cases - Homa



```
int homa_send(...)
{
        ...
        hdr.msg_control = &args;
        hdr.msg_controllen = 0;
        result = sendmsg(sockfd, &hdr, 0);
        ...
}
```



```
static int ____sys_sendmsg(...)
{
    unsigned char ctl[sizeof(struct cmsghdr) + 20]
                            __aligned(sizeof(__kernel_size_t));
    if (msg_sys->msg_controllen) {
        msg_sys->msg_control = ctl_buf;
        msg_sys->msg_control_is_user = false;
    }
    ...
}
```

Summary

- A lightweight zerocopy notification mechanism to save the overhead of extra system calls.

- A generic msg_control copy back framework in sendmsg, potentially apply to any other use cases where we need to return info back to user space.

Patchset

https://lore.kernel.org/all/20240708210405.870930-1-zijianzhang@bytedance.com/

Thanks for the code reviewing and suggestions by Willem de Bruijn and Xiaochun Lu!

Any comments or questions?

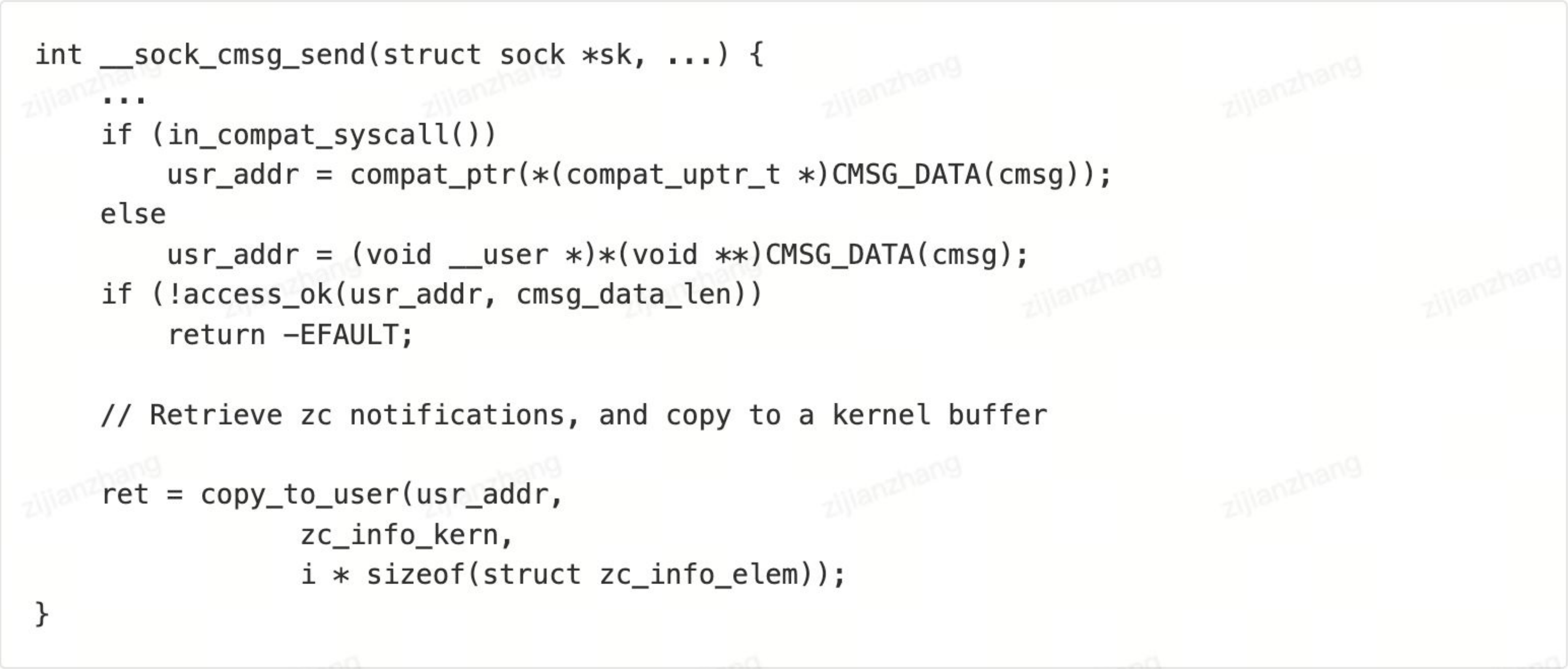
Patchset

https://lore.kernel.org/all/20240708210405.870930-1-zijianzhang@bytedance.com/

Thanks for the code reviewing and suggestions by Willem de Bruijn and Xiaochun Lu!

Open Question: Is this feature worth the minor cost in the sendmsg hot path?

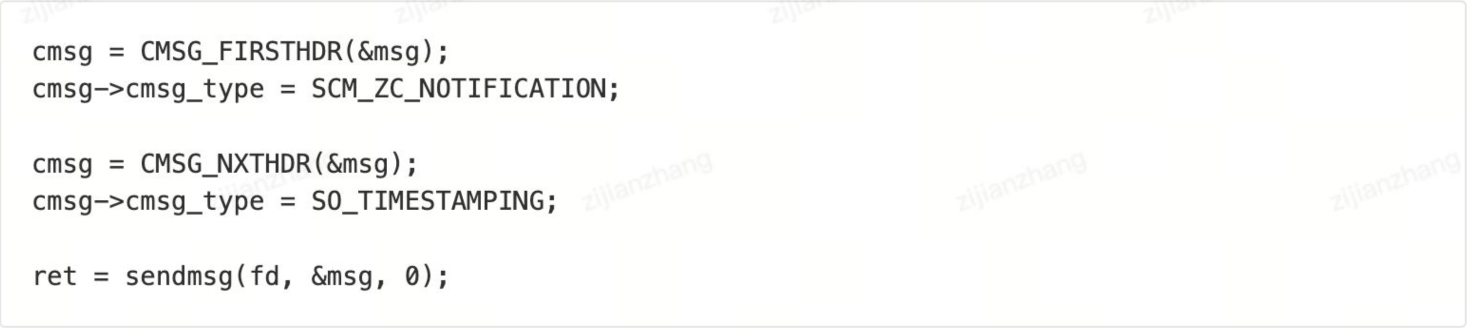
Appendix - design history



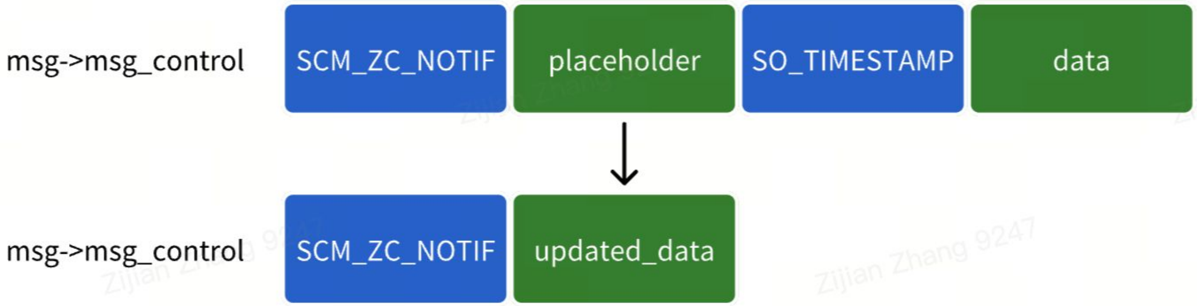
```
int __sock_cmsg_send(struct sock *sk, ...) {
    ...
    if (in_compat_syscall())
        usr_addr = compat_ptr(*(compat_uptr_t *)CMSG_DATA(cmsg));
    else
        usr_addr = (void __user *)*(void **)CMSG_DATA(cmsg);
    if (!access_ok(usr_addr, cmsg_data_len))
        return -EFAULT;

    // Retrieve zc notifications, and copy to a kernel buffer

    ret = copy_to_user(usr_addr,
                zc_info_kern,
                i * sizeof(struct zc_info_elem));
}
```

Reference: https://lore.kernel.org/all/20240528212103.350767-3-zijianzhang@bytedance.com/

Appendix - compatibility



```
cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_type = SCM_ZC_NOTIFICATION;

cmsg = CMSG_NXTHDR(&msg);
cmsg->cmsg_type = SO_TIMESTAMPING;

ret = sendmsg(fd, &msg, 0);
```



Background - history of TCP zerocopy in Linux: RX

The MSG_ZEROCOPY feature added in 4.14 enables zero-copy transmission of data, but does not address the receive side of the equation.

In linux-4.18, Eric Dumazet added a zero-copy receive mechanism to close that gap, at least for some relatively specialized applications.

- mmap() is used to reserve VMA space. tcp_mmap() makes sure pages will be Read Only.
- getsockopt(fd, IPPROTO_TCP, TCP_ZEROCOPY_RECEIVE, &zc, &zc_len); To implement actual mapping of pages into user space.

Reference: https://netdevconf.info//0x14/pub/slides/62

Background - history of TCP zerocopy in Linux: RX

Hardware features needed to support RX zero copy are limited to header split.

The size of the payload should be page-sized and page-aligned.

Reference: https://netdevconf.info//0x14/pub/slides/62