Linux Kernel Network Security (1)

Deep Hacking

2024/06/29 Pumpkin 🎃

Outline

- Overview
- AF_UNIX
- TLS
- BPF
- Others

Outline

- Overview
- AF_UNIX
- TLS
- BPF
- Others

Overview

In Linux 6.6.32

Overview

Socket Programming

```c
int main()
{
    int sockfd;
    struct sockaddr_in saddr;
    struct sockaddr_in caddr;
    char buf[0x10];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(1234);
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);

    bind(sockfd, (struct sockaddr*)&saddr, sizeof(saddr));
    listen(sockfd, 1);

    uint len = sizeof(caddr);
    int client = accept(sockfd, (struct sockaddr*)&caddr, &len);

    read(client, buf, sizeof(buf));
    write(client, buf, sizeof(buf));

    close(client);
    close(sockfd);
    return 0;
}
```

Server

```c
int main()
{
    int sockfd;
    struct sockaddr_in addr;
    char buf[0x10];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(1234);
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);

    connect(sockfd, (struct sockaddr*)&addr, sizeof(addr));

    write(sockfd, buf, sizeof(buf));
    read(sockfd, buf, sizeof(buf));

    close(sockfd);
    return 0;
}
```

Client

Overview

sys_socket

- int socket(int family, int type, int protocol);

Overview

sys_socket

- int socket(int family, int type, int protocol);
  - 檢查 type & family

```c
enum sock_type {
    SOCK_STREAM = 1,
    SOCK_DGRAM  = 2,
    SOCK_RAW    = 3,
    SOCK_RDM    = 4,
    SOCK_SEQPACKET  = 5,
    SOCK_DCCP   = 6,
    SOCK_PACKET = 10,
};
```

Type

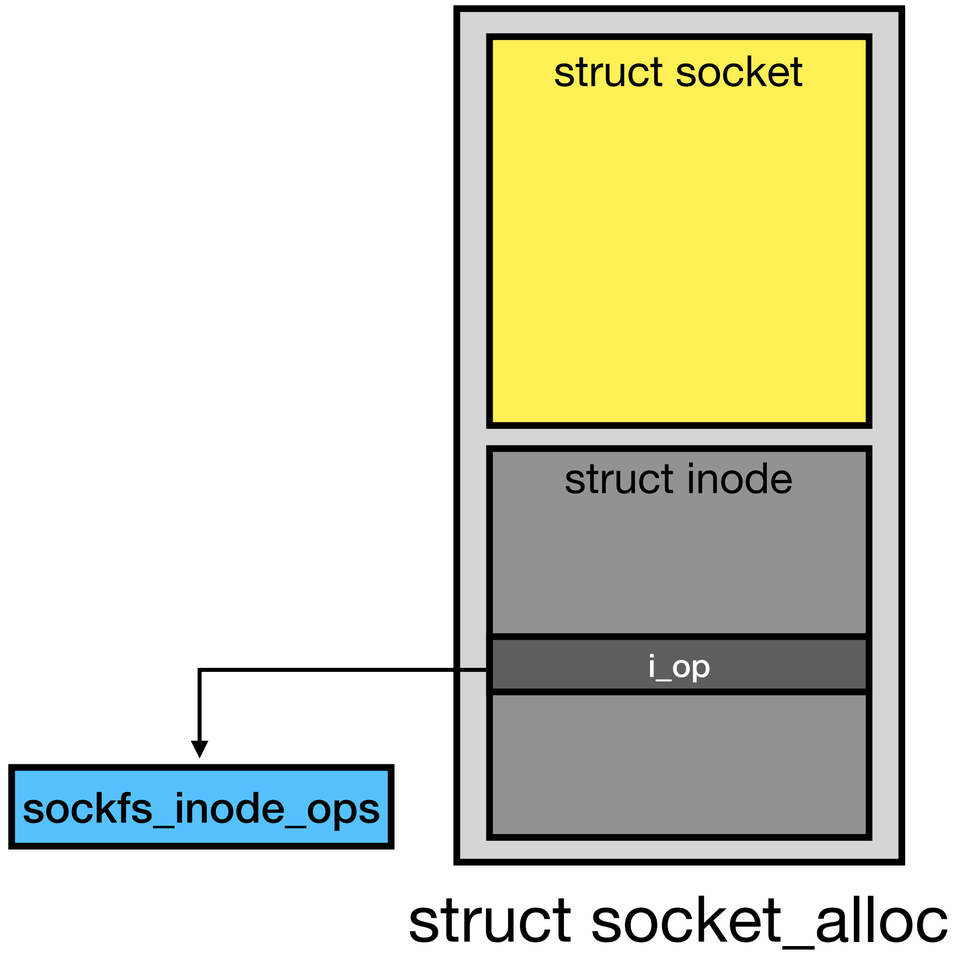
```c
#define AF_UNSPEC   0
#define AF_UNIX     1   /* Unix domain sockets      */
#define AF_LOCAL    1   /* POSIX name for AF_UNIX   */
#define AF_INET     2   /* Internet IP Protocol     */
// [...]
#define AF_XDP      44  /* XDP sockets          */
#define AF_MCTP     45  /* Management component
                         * transport protocol
                         */

#define AF_MAX      46  /* For now.. */
```

Family

Overview

sys_socket

- int socket(int family, int type, int protocol);
  - 檢查 type & family
  - `sock_alloc()`
    - inode = `alloc_inode()`
    - inode->i_op = &sockfs_inode_ops;

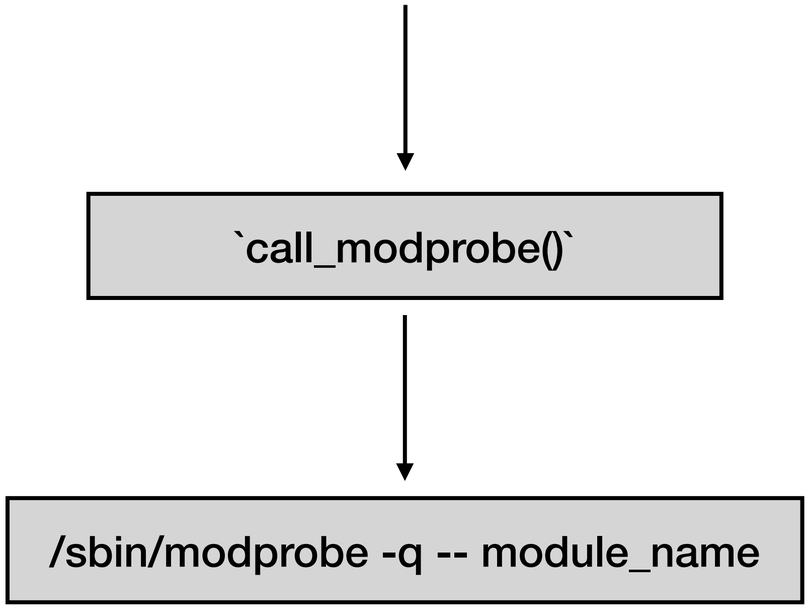


Overview

sys_socket

- int socket(int family, int type, int protocol);
  - 檢查 type & family
  - `sock_alloc()`
  - 嘗試載入 “net-pf-<family>“ kernel module

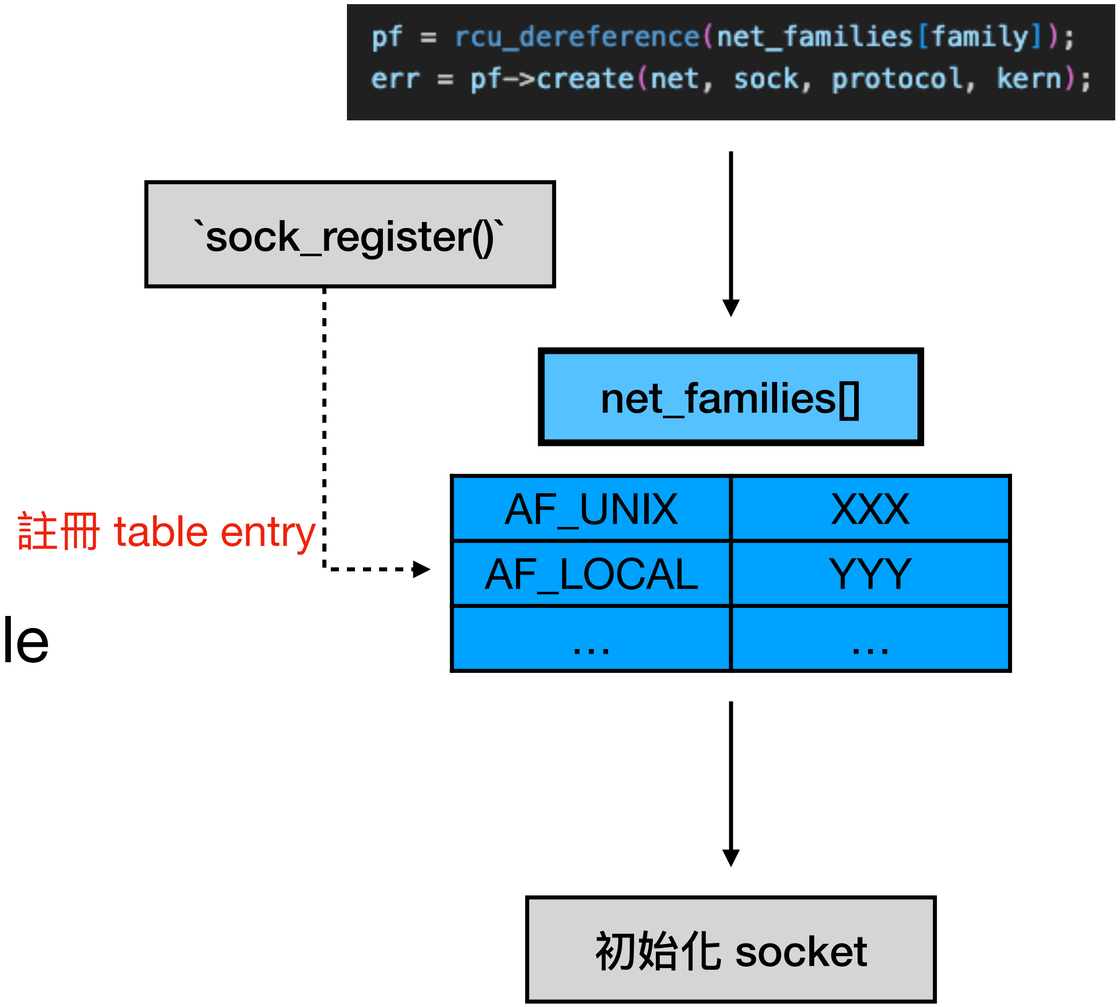
```
if (rcu_access_pointer(net_families[family]) == NULL)
    request_module("net-pf-%d", family);
```



Overview

sys_socket

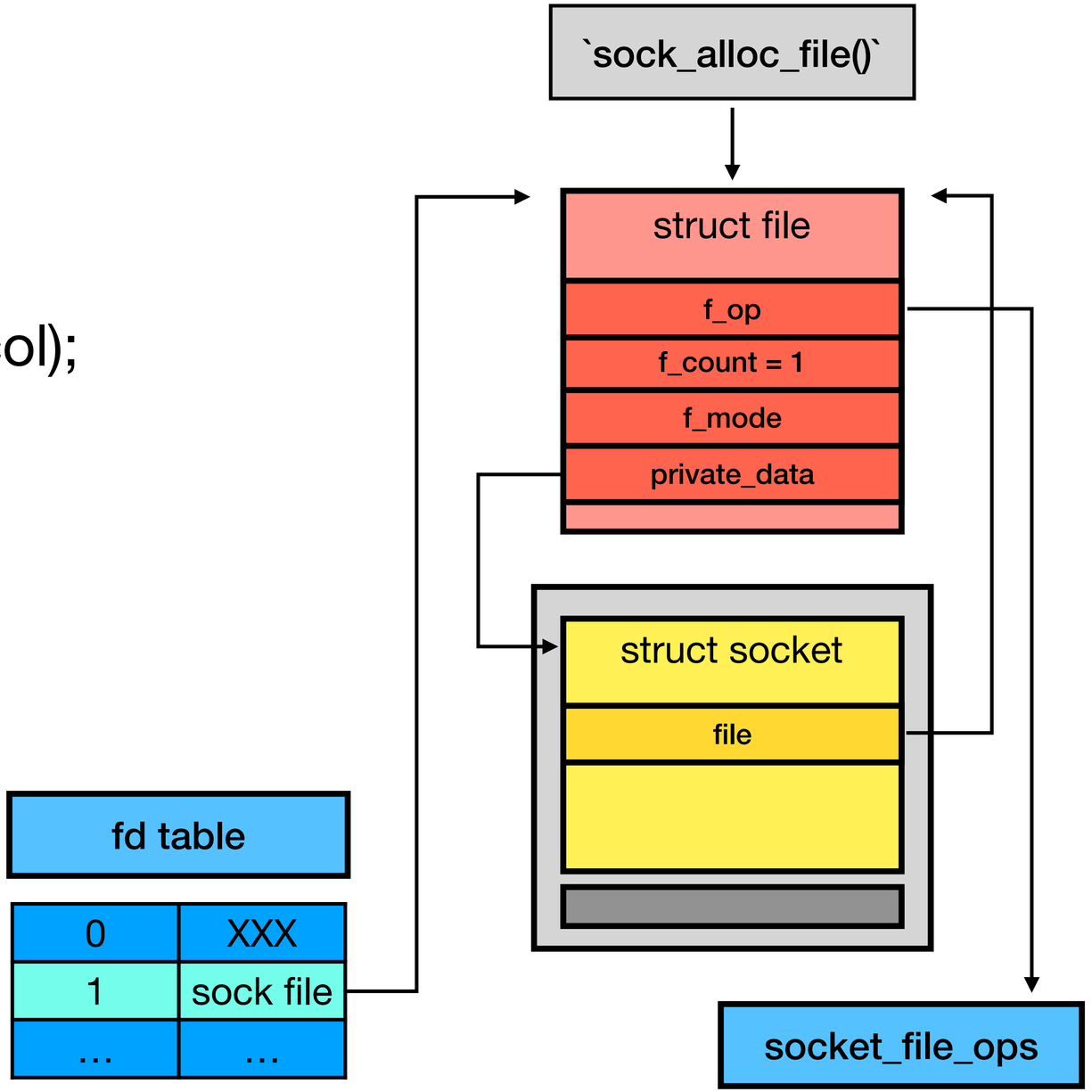
- int socket(int family, int type, int protocol);
  - 檢查 type & family
  - `sock_alloc()`
  - 嘗試載入 “net-pf-<family>“ kernel module
  - 呼叫對應 family 的 initializer



Overview

sys_socket

- int socket(int family, int type, int protocol);
  - …
  - Bind socket file 到 fd 上
    - newfile = `sock_alloc_file()`
    - Install newfile to fd table



Overview

sys_socket

- Family initializer (以 AF_INET 為例)
  - 檢查 protocol

```
enum {
  IPPROTO_IP = 0,    /* Dummy protocol for TCP   */
#define IPPROTO_IP     IPPROTO_IP
  IPPROTO_ICMP = 1,    /* Internet Control Message Protocol  */
  // [...]
#define IPPROTO_RAW    IPPROTO_RAW
  IPPROTO_MPTCP = 262,     /* Multipath TCP connection   */
#define IPPROTO_MPTCP    IPPROTO_MPTCP
  IPPROTO_MAX
};
```

Protocol

Overview

sys_socket

- Family initializer (以 AF_INET 為例)
  - 檢查 protocol
  - 嘗試載入對應 family-type-protocol 的 kernel module

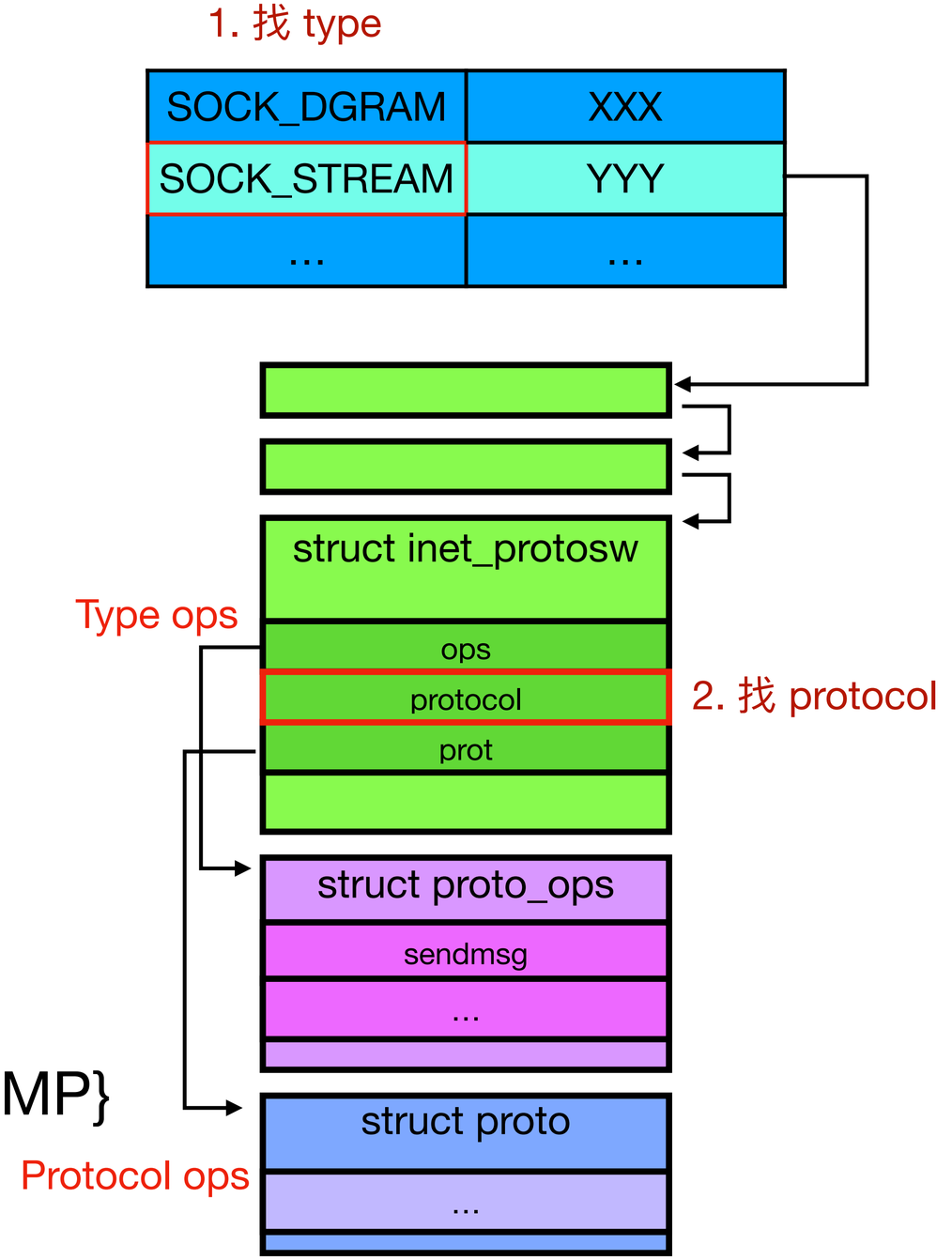
```
/*
 * Be more specific, e.g. net-pf-2-proto-132-type-1
 * (net-pf-PF_INET-proto-IPPROTO_SCTP-type-SOCK_STREAM)
 */
if (++try_loading_module == 1)
    request_module("net-pf-%d-proto-%d-type-%d",
                   PF_INET, protocol, sock->type);
/*
 * Fall back to generic, e.g. net-pf-2-proto-132
 * (net-pf-PF_INET-proto-IPPROTO_SCTP)
 */
else
    request_module("net-pf-%d-proto-%d",
                   PF_INET, protocol);
```

註解有範例

Overview

sys_socket

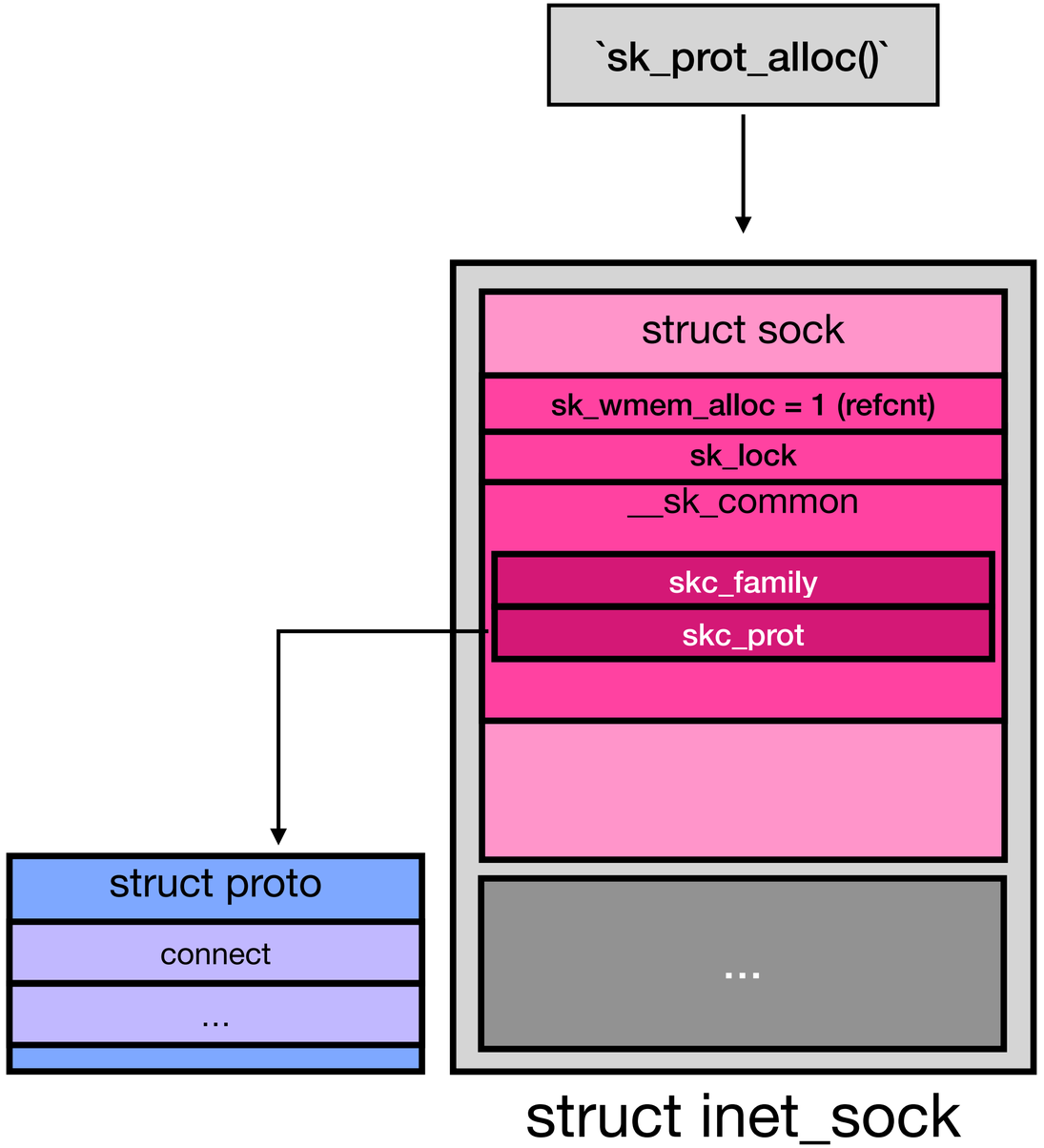
- Family initializer (以 AF_INET 為例)
  - 檢查 protocol
  - 嘗試載入對應 family-type-protocol 的 kernel module
  - 取得對應 type-protocol 的 operation table
    - 同個 type 可能會有多個 protocol,
      e.g. SOCK_DGRAM 有 IPPROTO_{UDP,ICMP}



Overview

sys_socket

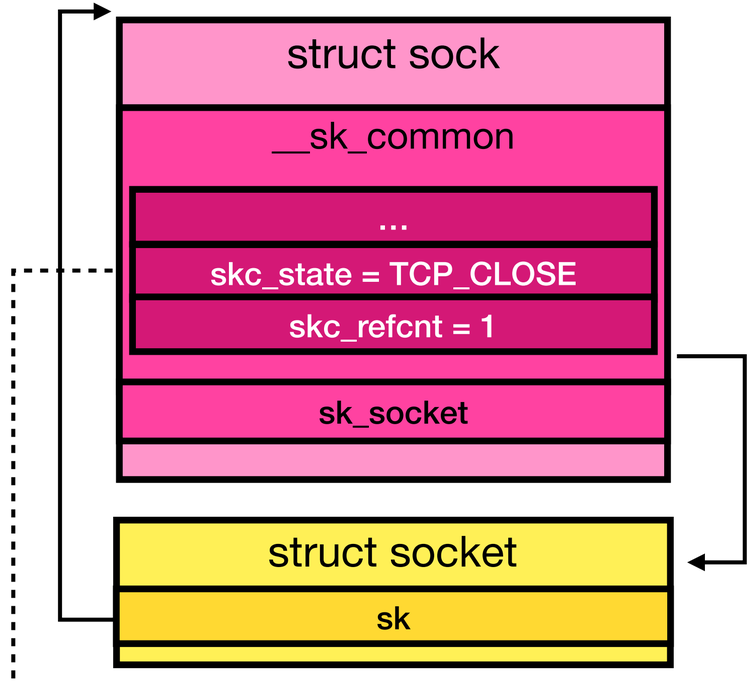
- Family initializer (以 AF_INET 為例)
  - …
  - `sk_alloc()`
    - `sk_prot_alloc()` 從 prot->slab 分配 sock object



Overview

sys_socket

- Family initializer (以 AF_INET 為例)
  - …
  - `sk_alloc()`
  - 初始化 sock
    - 一開始的 socket state 為 CLOSE



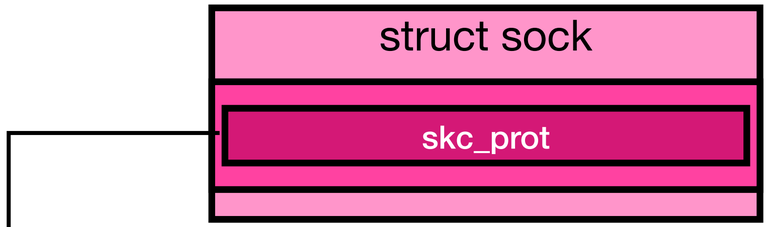

```
enum {
    TCP_ESTABLISHED = 1,
    TCP_SYN_SENT,
    TCP_SYN_RECV,
    // [...]
    TCP_CLOSE,
    // [...]
    TCP_NEW_SYN_RECV,
    TCP_MAX_STATES
};
```

Overview

sys_socket

- Family initializer (以 AF_INET 為例)
  - …
  - `sk_alloc()`
  - 初始化 sock
  - `sk->sk_prot->init()` protocol init sock

```
if (sk->sk_prot->init) {
    err = sk->sk_prot->init(sk);
    if (err) {
        sk_common_release(sk);
        goto out;
    }
}
```




```
struct proto tcp_prot = {
    .name           = "TCP",
    .owner          = THIS_MODULE,
    // [...]
    .init           = tcp_v4_init_sock,
    .destroy        = tcp_v4_destroy_sock,
    // [...]
```

Overview

sys_socket

- Protocol initializer (以 TCP 為例)
  - [1] 初始化與 TCP 連線相關的資料
  - [2] `icsk->icsk_af_ops = &ipv4_specific;`
    - TCP connection callback handler

```c
void tcp_init_sock(struct sock *sk)
{
    struct inet_connection_sock *icsk = inet_csk(sk);
    struct tcp_sock *tp = tcp_sk(sk);

    tp->out_of_order_queue = RB_ROOT;
    sk->tcp_rtx_queue = RB_ROOT;
    tcp_init_xmit_timers(sk);
    INIT_LIST_HEAD(&tp->tsq_node);
    INIT_LIST_HEAD(&tp->tsorted_sent_queue);

    icsk->icsk_rto = TCP_TIMEOUT_INIT;
    icsk->icsk_rto_min = TCP_RTO_MIN;
    icsk->icsk_delack_max = TCP_DELACK_MAX;
    tp->mdev_us = jiffies_to_usecs(TCP_TIMEOUT_INIT);
    minmax_reset(&tp->rtt_min, tcp_jiffies32, ~0U);
    // [...]
```

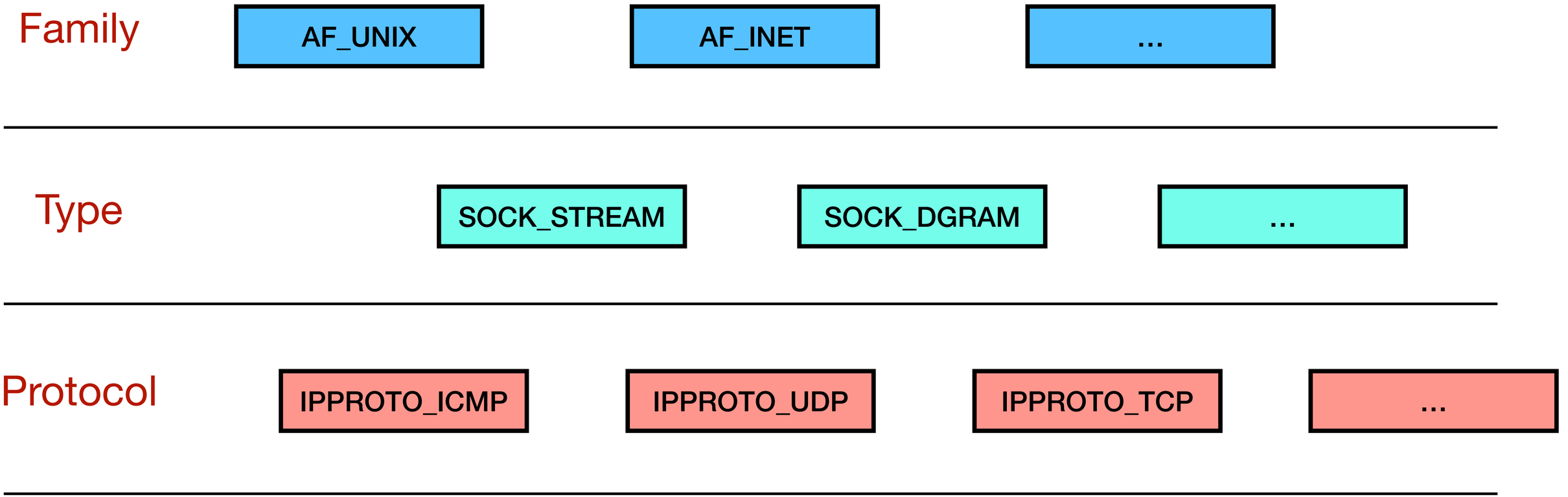
[1]

```c
const struct inet_connection_sock_af_ops ipv4_specific = {
    .queue_xmit     = ip_queue_xmit,
    .send_check     = tcp_v4_send_check,
    .rebuild_header     = inet_sk_rebuild_header,
    .sk_rx_dst_set      = inet_sk_rx_dst_set,
    .conn_request       = tcp_v4_conn_request,
    .syn_recv_sock      = tcp_v4_syn_recv_sock,
    .net_header_len     = sizeof(struct iphdr),
    .setsockopt     = ip_setsockopt,
    .getsockopt     = ip_getsockopt,
    .addr2sockaddr      = inet_csk_addr2sockaddr,
    .sockaddr_len       = sizeof(struct sockaddr_in),
    .mtu_reduced        = tcp_v4_mtu_reduced,
};
```

[2]

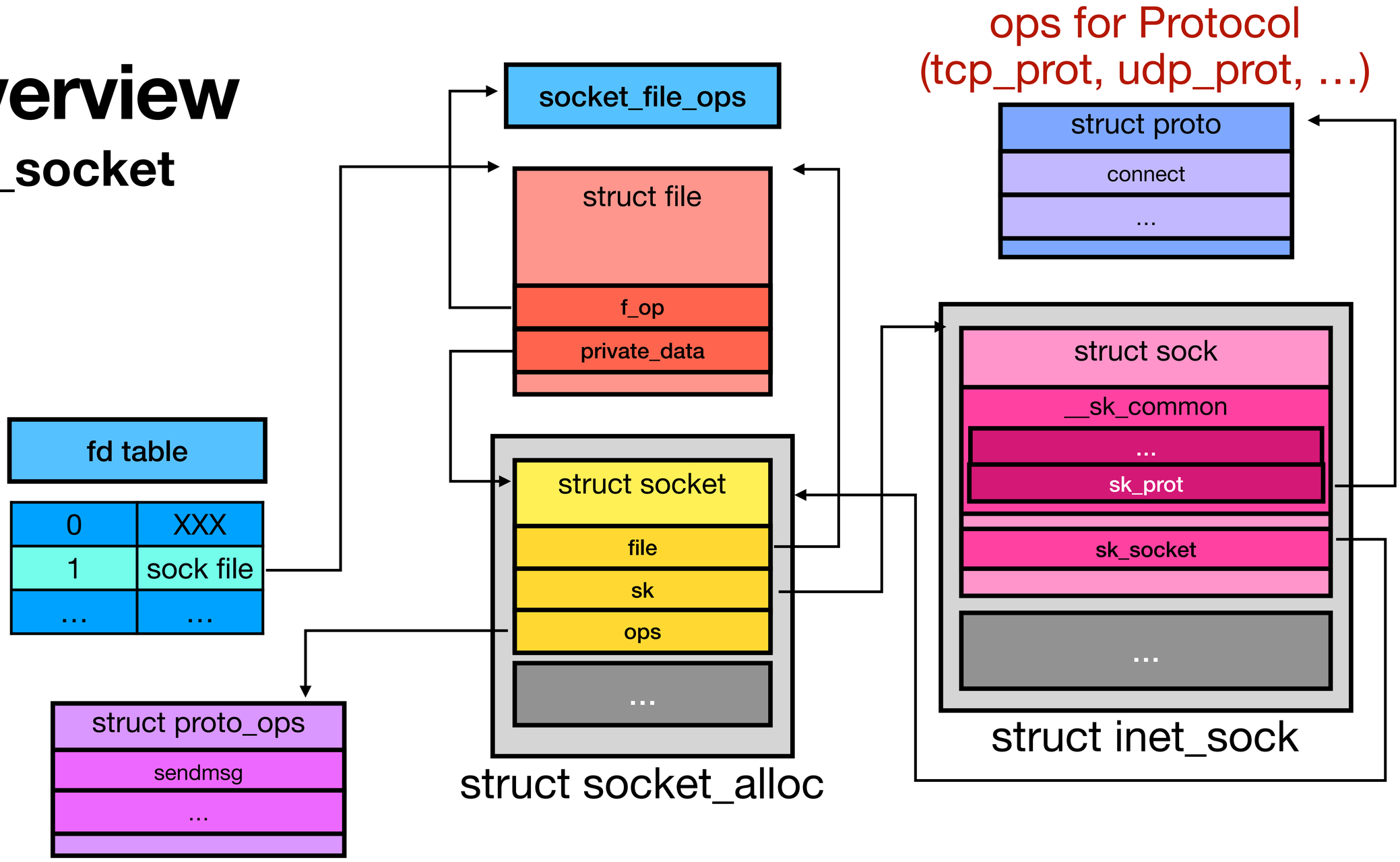
Overview

sys_socket



Overview

sys_socket



Overview

sys_bind

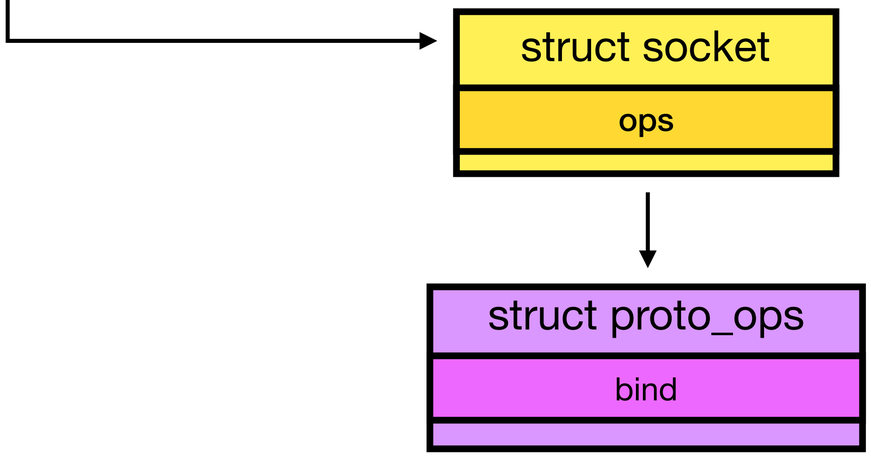
- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Overview

sys_bind

- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  - `sockfd_lookup_light()`
    - `fdget()` 拿 file
    - `sock_from_file()` 拿 socket
  - Call sock->ops->bind()
    - Type bind handler

```c
static struct socket *sockfd_lookup_light(int fd, int *err, int *fput_needed)
{
    struct fd f = fdget(fd);
    struct socket *sock;

    sock = sock_from_file(f.file);
    *fput_needed = f.flags & FDPUT_FPUT;
    return sock;
}
```



Overview

sys_bind

- Type bind handler (以 inet_stream_ops 為例)
  - [1] 如果有 protocol bind handler 就用 (sk->sk_prot->bind)
  - [2] 檢查 socket 狀態是否為 TCP_CLOSE
  - [3] 更新 address 與 port

```c
/* If the socket has its own bind function then use it. (RAW) */
if (sk->sk_prot->bind) {
    return sk->sk_prot->bind(sk, uaddr, addr_len);
}
```

[1]

```c
/* Check these errors (active socket, double bind). */
err = -EINVAL;
if (sk->sk_state != TCP_CLOSE || inet->inet_num)
    goto out_release_sock;
```

[2]

```c
inet->inet_sport = htons(inet->inet_num);
inet->inet_daddr = 0;
inet->inet_dport = 0;
sk_dst_reset(sk);
err = 0;
```

[3]

Overview

sys_listen

- int listen(int sockfd, int backlog);

Overview

sys_listen

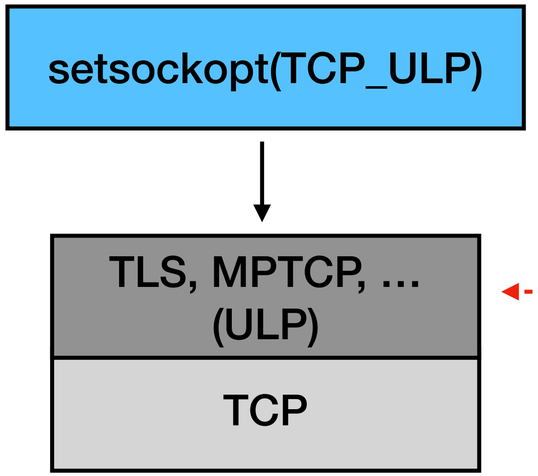
- int listen(int sockfd, int backlog);
  - `sockfd_lookup_light()`
  - Call `sock->ops->listen()`
    - Type listen handler

```c
int __sys_listen(int fd, int backlog)
{
    struct socket *sock;
    int err, fput_needed;

    sock = sockfd_lookup_light(fd, &err, &fput_needed);
    err = READ_ONCE(sock->ops)->listen(sock, backlog);
    fput_light(sock->file, fput_needed);

    return err;
}
```

Overview

sys_listen

- Type listen handler
  - [1] Take sock ownership
  - [2] LISTEN mode 初始化
    - Allocate accept queue
    - Update state to LISTEN
  - [3] Release sock ownership




```c
int inet_csk_listen_start(struct sock *sk)
{
    struct inet_connection_sock *icsk = inet_csk(sk);
    struct inet_sock *inet = inet_sk(sk);
    int err;

    err = inet_ulp_can_listen(sk);
    // [...]
    reqsk_queue_alloc(&icsk->icsk_accept_queue);
    // [...]
    inet_sk_state_store(sk, TCP_LISTEN);
    // [...]
    return 0;
```

[2]

Overview

sys_accept

- int accept(int sockfd, struct sockaddr *_Nullable restrict addr, socklen_t *_Nullable restrict addrlen);

Overview

sys_accept

- int accept(int sockfd, struct sockaddr *_Nullable restrict addr, socklen_t *_Nullable restrict addrlen);
  - newfile = `do_accept()`
    - 分配新的 socket, file object
    - 繼承 parent socket 的 type, ops, …
    - Call `sock->ops->accept()`
      - Type accept handler

```
struct file *do_accept(struct file *file, unsigned file_flags,
                       struct sockaddr __user *upeer_sockaddr,
                       int __user *upeer_addrlen, int flags)
{
    struct socket *sock, *newsock;
    struct file *newfile;
    int err, len;
    struct sockaddr_storage address;
    const struct proto_ops *ops;

    sock = sock_from_file(file);
    newsock = sock_alloc();
    ops = READ_ONCE(sock->ops);
    newsock->type = sock->type;
    newsock->ops = ops;

    newfile = sock_alloc_file(newsock, flags, sock->sk->sk_prot_creator->name);
    err = ops->accept(sock, newsock, sock->file->f_flags | file_flags,
                      false);
```

Overview

sys_accept

- int accept(int sockfd, struct sockaddr *_Nullable restrict addr, socklen_t *_Nullable restrict addrlen);
  - …
  - Install newfile to fd table

```
newfd = get_unused_fd_flags(flags);

newfile = do_accept(file, 0, upeer_sockaddr, upeer_addrlen,
                    flags);

fd_install(newfd, newfile);
```

Overview

sys_accept

- Type accept handler
  - Call `sk->sk_prot->accept()`
    - Protocol accept handler
  - Take sock ownership
  - `__inet_accept()`
  - Release sock ownership

```
int inet_accept(struct socket *sock, struct socket *newsock, int flags,
        bool kern)
{
    struct sock *sk1 = sock->sk, *sk2;
    int err = -EINVAL;

    sk2 = READ_ONCE(sk1->sk_prot)->accept(sk1, flags, &err, kern);
    lock_sock(sk2);
    __inet_accept(sock, newsock, sk2);
    release_sock(sk2);
    return 0;
}
```

Overview

sys_accept

- Type accept handler
  - `__inet_accept()`
    - Bind new sock to new socket
    - Set new socket state to CONNECTED

```
void __inet_accept(struct socket *sock, struct socket *newsock, struct sock *newsk)
{
    // [...]
    sock_graft(newsk, newsock);
    newsock->state = SS_CONNECTED;
}
```

Overview

sys_accept

- Protocol accept handler (以 tcp_prot 為例)
  - 檢查 state 是否為 LISTEN
  - Wait for connection
  - Get new sock object from connection request
  - Init lock of new sock

```
struct sock *inet_csk_accept(struct sock *sk, int flags, int *err
{
    struct inet_connection_sock *icsk = inet_csk(sk);
    struct request_sock_queue *queue = &icsk->icsk_accept_queue;
    struct request_sock *req;
    struct sock *newsk;
    int error;

    // [...]
    if (sk->sk_state != TCP_LISTEN)
        goto out_err;

    if (reqsk_queue_empty(queue)) {
        long timeo = sock_rcvtimeo(sk, flags & O_NONBLOCK);
        error = inet_csk_wait_for_connect(sk, timeo);
    }
    req = reqsk_queue_remove(queue, sk);
    newsk = req->sk;
    // [...]
    reqsk_put(req);
    inet_init_csk_locks(newsk);

    return newsk;
```

Overview

sys_connect

- int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Overview

sys_connect

- int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  - `fdget()` 拿 file
  - `sock_from_file()` 拿 socket
  - Call `sock->ops->connect()`
    - Type connect handler

```
int __sys_connect_file(struct file *file, struct sockaddr_storage *address,
                       int addrlen, int file_flags)
{
    struct socket *sock;
    int err;

    sock = sock_from_file(file);
    err = READ_ONCE(sock->ops)->connect(sock, (struct sockaddr *)address,
                        addrlen, sock->file->f_flags | file_flags);
out:
    return err;
}
```

Overview

sys_connect

- Type connect handler
  - Take sock ownership
  - `__inet_stream_connect()`
  - Release sock ownership

```c
int inet_stream_connect(struct socket *sock, struct sockaddr *uaddr,
                        int addr_len, int flags)
{
    int err;

    lock_sock(sock->sk);
    err = __inet_stream_connect(sock, uaddr, addr_len, flags, 0);
    release_sock(sock->sk);
    return err;
}
```

Overview

sys_connect

- Type connect handler
  - `__inet_stream_connect()`
    - 檢查 socket state 為 UNCONNECTED，sock state 為 TCP_CLOSE
    - Call `sk->sk_prot->connect()`
      - Protocol accept handler
    - Update socket state to CONNECTING

```
case SS_UNCONNECTED:
    err = -EISCONN;
    if (sk->sk_state != TCP_CLOSE)
        goto out;

    err = sk->sk_prot->connect(sk, uaddr, addr_len);
    if (err < 0)
        goto out;

    sock->state = SS_CONNECTING;
```

Overview

sys_connect

- Type connect handler
  - `__inet_stream_connect()`
    - …
    - `inet_wait_for_connect()` TCP handshaking
      - 等待 server 的回覆
      - 連線成功後更新 sock state to ESTABLISHED
    - Update socket state to CONNECTED

```
// [...]
if (!timeo || !inet_wait_for_connect(sk, timeo, writebias))
    goto out;
// [...]

if (sk->sk_state == TCP_CLOSE)
    goto sock_error;

sock->state = SS_CONNECTED;
```

```
void tcp_finish_connect(struct sock *sk, struct sk_buff *skb)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct inet_connection_sock *icsk = inet_csk(sk);

    tcp_set_state(sk, TCP_ESTABLISHED);
```

Overview

sys_connect

- Protocol connect handler
  - [1] Find routing rule
  - [2] Setup sock and set state to TCP_SYN_SENT
  - [3] `tcp_connect()`

```
orig_sport = inet->inet_sport;
orig_dport = usin->sin_port;
fl4 = &inet->cork.fl.u.ip4;
rt = ip_route_connect(fl4, nexthop, inet->inet_saddr,
                      sk->sk_bound_dev_if, IPPROTO_TCP, orig_sport,
                      orig_dport, sk);
```

[1]

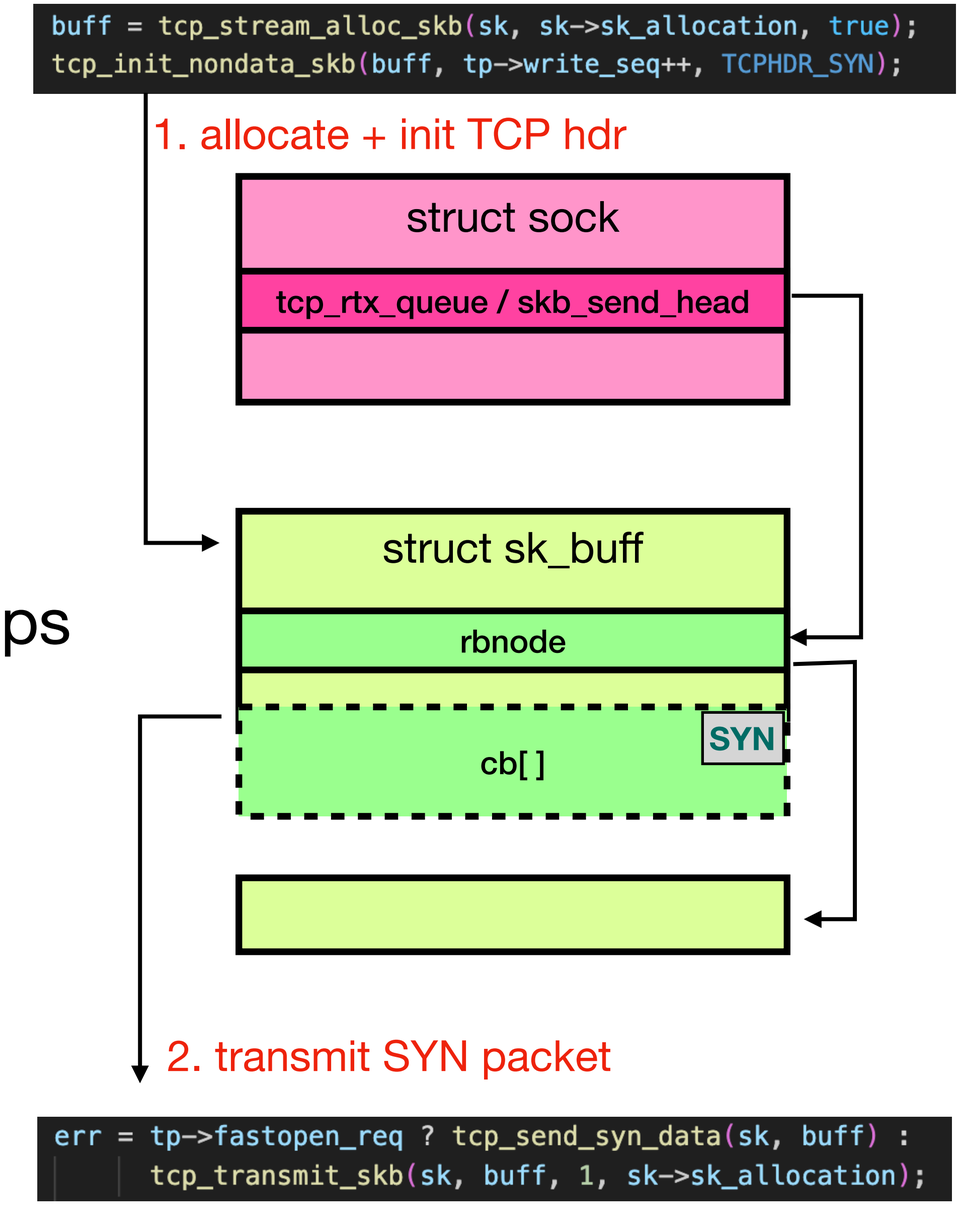
```
tcp_set_state(sk, TCP_SYN_SENT);
```

[2]

Overview

sys_connect

- Protocol connect handler
  - `tcp_connect()`
    - [1] `tcp_connect_init(sk)` connect socket setups
    - [2] allocate SYN sk_buff
    - [3] 發送 SYN sk_buff packet



Overview

sys_read

- ssize_t read(int fd, void buf[.count], size_t count);

Overview

sys_read

- ssize_t read(int fd, void buf[.count], size_t count);
  - Verify read address and size
  - Call `file->f_op->read_iter()`
    - f_op 為 socket_file_ops
    - Read handler 為 sock_read_iter
      - 最後走到 `sock->ops->recvmsg()`

```
static const struct file_operations socket_file_ops = {
    .owner =    THIS_MODULE,
    .llseek =   no_llseek,
    .read_iter =    sock_read_iter,
    .write_iter =   sock_write_iter,
    .poll =     sock_poll,
    .unlocked_ioctl = sock_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = compat_sock_ioctl,
#endif
    .uring_cmd =    io_uring_cmd_sock,
    .mmap =     sock_mmap,
    .release =  sock_close,
    .fasync =   sock_fasync,
    .splice_write = splice_to_socket,
    .splice_read =  sock_splice_read,
    .splice_eof =   sock_splice_eof,
    .show_fdinfo =  sock_show_fdinfo,
};
```

```
static inline int sock_recvmsg_nosec(struct socket *sock, struct msghdr *msg,
                                     int flags)
{
    int ret = INDIRECT_CALL_INET(READ_ONCE(sock->ops)->recvmsg,
                                 inet6_recvmsg,
                                 inet_recvmsg, sock, msg,
                                 msg_data_left(msg), flags);
```

Overview

sys_read

- Type recvmsg handler
  - [1] Call protocol recvmsg handler
- Protocol recvmsg handler
  - [2] Take sock ownership
  - [3] `tcp_recvmsg_locked()`
  - [4] Release sock ownership

```
err = INDIRECT_CALL_2(sk->sk_prot->recvmsg, tcp_recvmsg, udp_recvmsg,
                      sk, msg, size, flags, &addr_len);
```

[1]

```
lock_sock(sk);
ret = tcp_recvmsg_locked(sk, msg, len, flags, &tss, &cmsg_flags);
release_sock(sk);
```

[2], [3], [4]

Overview

sys_read

- Protocol recvmsg handler
  - `tcp_recvmsg_locked()`
    - [1] Walk the receive_queue
    - [2] Copy datagram to msg
    - [3] Consume the used skb

```
last = skb_peek_tail(&sk->sk_receive_queue);
skb_queue_walk(&sk->sk_receive_queue, skb) {
    last = skb;
    // [...]
    offset = *seq - TCP_SKB_CB(skb)->seq;
    if (offset < skb->len)
        goto found_ok_skb;
```

[1]

```
if (!(flags & MSG_TRUNC)) {
    err = skb_copy_datagram_msg(skb, offset, msg, used);
```

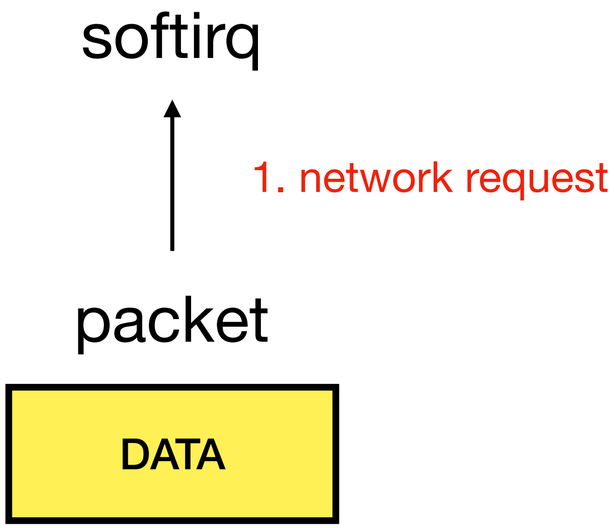
[2]

```
static void tcp_eat_recv_skb(struct sock *sk, struct sk_buff *skb)
{
    __skb_unlink(skb, &sk->sk_receive_queue);
    if (likely(skb->destructor == sock_rfree)) {
        sock_rfree(skb);
        skb->destructor = NULL;
        skb->sk = NULL;
        return skb_attempt_defer_free(skb);
    }
    __kfree_skb(skb);
}
```

[3]

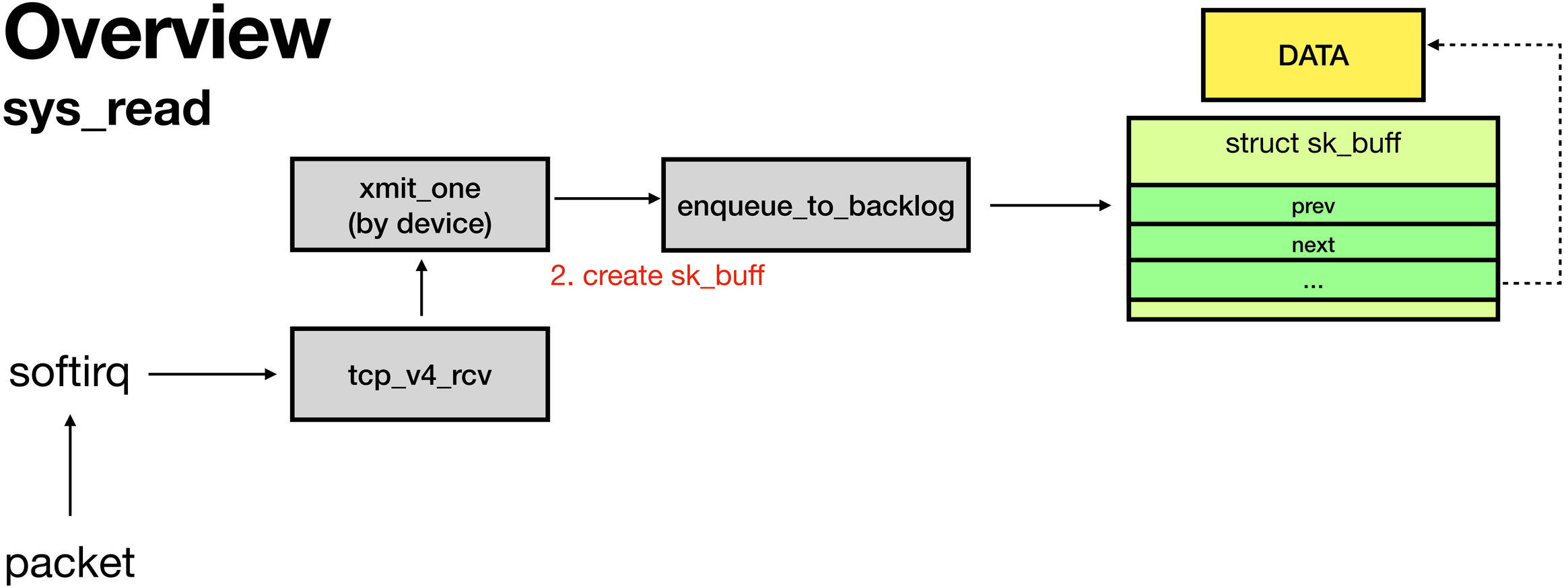
Overview

sys_read



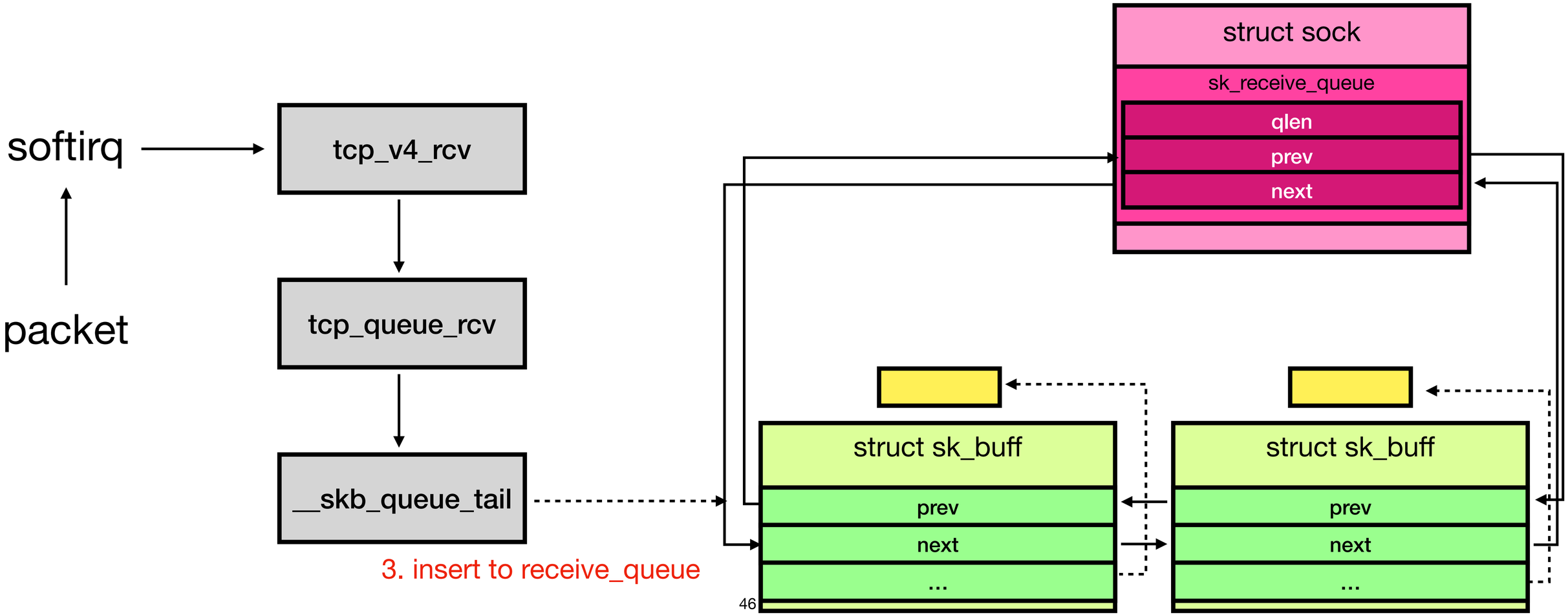
Overview

sys_read



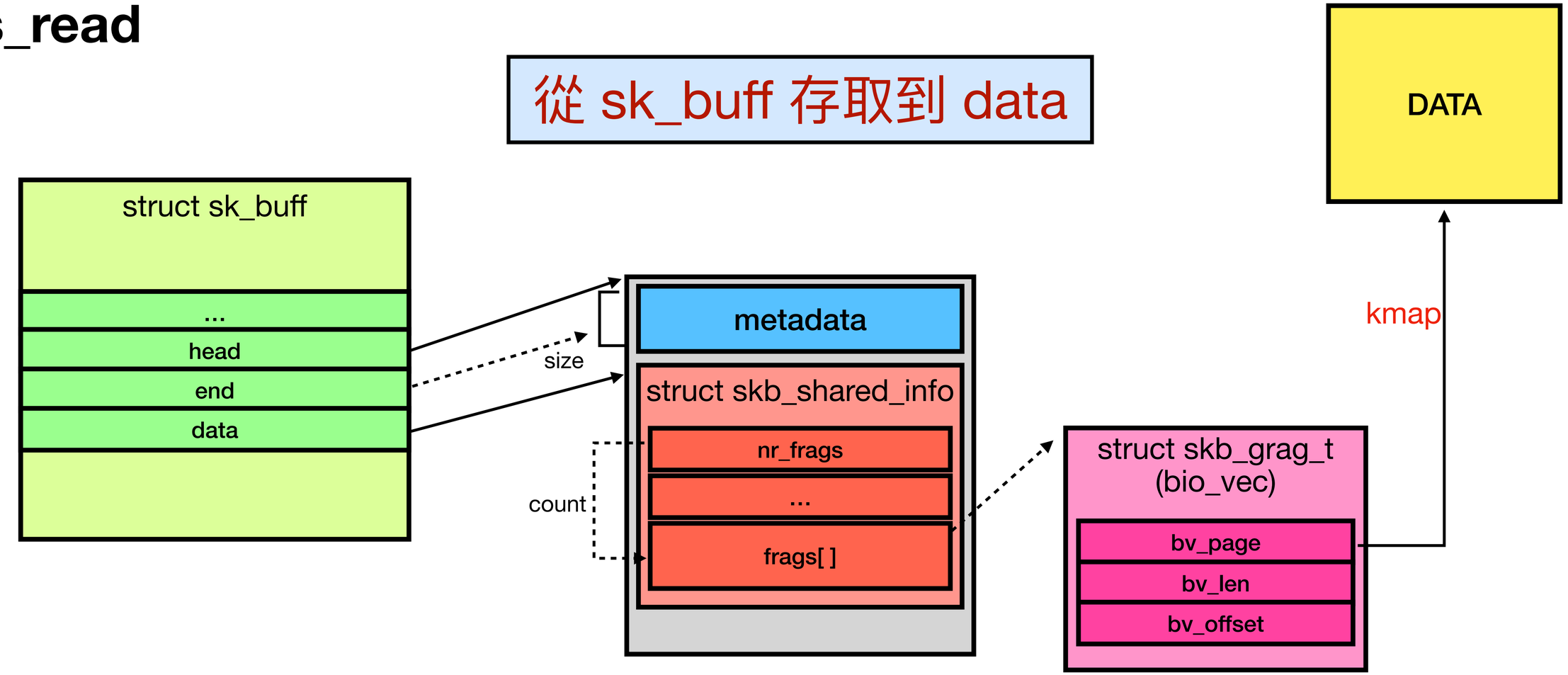
Overview

sys_read



Overview

sys_read



Overview

sys_{recv, recvfrom}

- ssize_t recv(int sockfd, void buf[.len], size_t len, int flags);
- ssize_t recvfrom(int sockfd, void buf[restrict .len], size_t len, int flags, struct sockaddr *_Nullable restrict src_addr, socklen_t *_Nullable restrict addrlen);

```
SYSCALL_DEFINE6(recvfrom, int, fd, void __user *, ubuf, size_t, size,
        unsigned int, flags, struct sockaddr __user *, addr,
        int __user *, addr_len)
{
    return __sys_recvfrom(fd, ubuf, size, flags, addr, addr_len);
}

SYSCALL_DEFINE4(recv, int, fd, void __user *, ubuf, size_t, size,
        unsigned int, flags)
{
    return __sys_recvfrom(fd, ubuf, size, flags, NULL, NULL);
}
```

Overview

sys_{recv, recvfrom}

- `__sys_recvfrom()`
  - `sockfd_lookup_light()`
  - `sock_recvmsg()`
    - Call type recvmsg handler，就跟 sys_read 一樣

```
static inline int sock_recvmsg_nosec(struct socket *sock, struct msghdr *msg,
                     int flags)
{
    int ret = INDIRECT_CALL_INET(READ_ONCE(sock->ops)->recvmsg,
                     inet6_recvmsg,
                     inet_recvmsg, sock, msg,
                     msg_data_left(msg), flags);
```

Overview

sys_recvmsg

- ssize_t recvmsg(int socket, struct msghdr *message, int flags);

Overview

sys_recvmsg

- ssize_t recvmsg(int socket, struct msghdr *message, int flags);
  - [1] `sockfd_lookup_light()`
  - [2] `___sys_recvmsg()`
    - [3] Setup msghdr
    - [4] `____sys_recvmsg()`
      - [5] `sock_recvmsg()`

```
static int ___sys_recvmsg(struct socket *sock, struct user_msghdr __user *msg,
                          struct msghdr *msg_sys, unsigned int flags, int nosec)
{
    struct iovec iovstack[UIO_FASTIOV], *iov = iovstack;
    /* user mode address pointers */
    struct sockaddr __user *uaddr;
    ssize_t err;

    err = recvmsg_copy_msghdr(msg_sys, msg, flags, &uaddr, &iov);
    err = ____sys_recvmsg(sock, msg_sys, msg, uaddr, flags, nosec);
```

[2], [3], [4]

```
msg_sys->msg_name = &addr;
msg_sys->msg_flags = flags & (MSG_CMSG_CLOEXEC|MSG_CMSG_COMPAT);
msg_sys->msg_namelen = 0;
// [...]
err = sock_recvmsg_nosec(sock, msg_sys, flags);
```

[5]

Overview

sys_write

- ssize_t write(int fd, const void buf[.count], size_t count);

Overview

sys_write

- ssize_t write(int fd, const void buf[.count], size_t count);
  - Verify write address and size
  - Call `file->f_op->write_iter()`
    - Write handler 為 sock_write_iter
      - 最後走到 `sock->ops->sendmsg()`

```
static ssize_t sock_write_iter(struct kiocb *iocb, struct iov_iter *from)
{
    struct file *file = iocb->ki_filp;
    struct socket *sock = file->private_data;
    struct msghdr msg = {.msg_iter = *from,
                         .msg_iocb = iocb};
    ssize_t res;
    // [...]
    res = __sock_sendmsg(sock, &msg);
```

```
static inline int sock_sendmsg_nosec(struct socket *sock, struct msghdr *msg)
{
    int ret = INDIRECT_CALL_INET(READ_ONCE(sock->ops)->sendmsg, inet6_sendmsg,
                                 inet_sendmsg, sock, msg,
                                 msg_data_left(msg));
    BUG_ON(ret == -EIOCBQUEUED);
```

Overview

sys_write

- Type sendmsg handler
  - [1] Call protocol sendmsg handler
- Protocol sendmsg handler
  - [2] Take sock ownership
  - [3] `tcp_sendmsg_locked()`
  - [4] Release sock ownership

```c
int tcp_sendmsg(struct sock *sk, struct msghdr *msg, size_t size)
{
    int ret;

    lock_sock(sk);
    ret = tcp_sendmsg_locked(sk, msg, size);
    release_sock(sk);
```

[2], [3], [4]

Overview

sys_write

- Protocol sendmsg handler
  - `tcp_sendmsg_locked()`
    - [1] Allocate sk_buff
    - [2] 資料寫到對應的 fragment
    - [3] move skb from write_queue to tcp_rtx_queue

```
skb = tcp_stream_alloc_skb(sk, sk->sk_allocation,
                           first_skb);
process_backlog++;

tcp_skb_entail(sk, skb);
```

[1]

```
int i = skb_shinfo(skb)->nr_frags;
struct page_frag *pfrag = sk_page_frag(sk);
// [...]
err = skb_copy_to_page_nocache(sk, &msg->msg_iter, skb,
                               pfrag->page,
                               pfrag->offset,
                               copy);
```

[2]

```
static void tcp_event_new_data_sent(struct sock *sk, struct sk_buff *skb)
{
    struct inet_connection_sock *icsk = inet_csk(sk);
    struct tcp_sock *tp = tcp_sk(sk);
    // [...]
    __skb_unlink(skb, &sk->sk_write_queue);
    tcp_rbtree_insert(&sk->tcp_rtx_queue, skb);
```

[3]

Overview

sys_{send, sendto}

- ssize_t send(int sockfd, const void buf[.len], size_t len, int flags);
- ssize_t sendto(int sockfd, const void buf[.len], size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

```
SYSCALL_DEFINE6(sendto, int, fd, void __user *, buff, size_t, len,
        unsigned int, flags, struct sockaddr __user *, addr,
        int, addr_len)
{
    return __sys_sendto(fd, buff, len, flags, addr, addr_len);
}

SYSCALL_DEFINE4(send, int, fd, void __user *, buff, size_t, len,
        unsigned int, flags)
{
    return __sys_sendto(fd, buff, len, flags, NULL, 0);
}
```

Overview

sys_{send, sendto}

- `__sys_sendto()`
  - `sockfd_lookup_light()`
  - `__sock_sendmsg()`
    - Call type sendmsg handler，就跟 sys_write 一樣

Overview

sys_sendmsg

- ssize_t sendmsg(int socket, const struct msghdr *message, int flags);

Overview

sys_sendmsg

- ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
  - [1] `sockfd_lookup_light()`
  - [2] `___sys_sendmsg()`
    - [3] Setup msghdr
    - [4] `____sys_sendmsg()`
      - [5] `__sock_sendmsg()`

```
err = sendmsg_copy_msghdr(msg_sys, msg, flags, &iov);
if (err < 0)
    return err;

err = ____sys_sendmsg(sock, msg_sys, flags, used_address,
                      allowed_msghdr_flags);
```

[3], [4]

```
err = __sock_sendmsg(sock, msg_sys);
```

[5]

Overview

sys_close

- int close(int fd);

Overview

sys_close

- int close(int fd);
  - `close_fd_get_file()`
    - [1] Files lock
    - [2] 根據 fd 從 fd table 拿 file，unset fdentry
    - [3] Files unlock
    - [4] Call` __fput_sync()`

```
spin_lock(&files->file_lock);
file = pick_file(files, fd);
spin_unlock(&files->file_lock);
```

[1], [3]

```
static struct file *pick_file(struct files_struct *files, unsigned fd)
{
    struct fdtable *fdt = files_fdtable(files);
    struct file *file;
    // [...]
    fd = array_index_nospec(fd, fdt->max_fds);
    file = fdt->fd[fd];
    if (file) {
        rcu_assign_pointer(fdt->fd[fd], NULL);
        __put_unused_fd(files, fd);
```

[2]

Overview

sys_close

- int close(int fd);
  - `__fput_sync()`
    - file's refcnt -= 1
    - 若結果為 0 就呼叫 `__fput()`

```
void __fput_sync(struct file *file)
{
    if (atomic_long_dec_and_test(&file->f_count))
        __fput(file);
}
```

Overview

sys_close

- int close(int fd);
  - `__fput()`
    - [1] Call `file->f_op->release()`
      - Release handler 為 sock_close
        - [2] `ops->release(sock)`
          - Call type release handler

```c
if (file->f_op->release)
    file->f_op->release(inode, file);
```

[1]

```c
if (ops) {
    struct module *owner = ops->owner;

    if (inode)
        inode_lock(inode);
    ops->release(sock);
    sock->sk = NULL;
    if (inode)
        inode_unlock(inode);
    sock->ops = NULL;
    module_put(owner);
}
```

[2]

Overview

sys_close

- int close(int fd);
  - `__fput()`
    - …
    - 釋放 dentry + file object
      - RCU call `file->f_op->free_inode()`
        - Free inode handler 為 sock_free_inode

```
dput(dentry);
// [...]
file_free(file);
```

[1]

```
static void sock_free_inode(struct inode *inode)
{
    struct socket_alloc *ei;

    ei = container_of(inode, struct socket_alloc, vfs_inode);
    kmem_cache_free(sock_inode_cachep, ei);
}
```

[2]

Overview

sys_close

- Type release handler
  - Call protocol close handler
- Protocol close handler
  - Take sock ownership
  - Close tcp connection
  - Release sock ownership
  - Call `sock_put()`

```
void tcp_close(struct sock *sk, long timeout)
{
    lock_sock(sk);
    __tcp_close(sk, timeout);
    release_sock(sk);
    sock_put(sk);
}
```

Overview

sys_close

- Protocol close handler
  - Close tcp connection
    - [1] Flush recv queue
    - [2] Send TCP FIN request
    - [3] Wait for the handshake to end

```
while ((skb = __skb_dequeue(&sk->sk_receive_queue)) != NULL) {
    u32 len = TCP_SKB_CB(skb)->end_seq - TCP_SKB_CB(skb)->seq;

    if (TCP_SKB_CB(skb)->tcp_flags & TCPHDR_FIN)
        len--;
    data_was_unread += len;
    __kfree_skb(skb);
}
```

[1]

```
    tcp_send_fin(sk);
}

sk_stream_wait_close(sk, timeout);
```

[2]

Overview

sys_close

- Protocol close handler
  - [1] `sock_put()`
    - sock's refcnt -= 1
    - 若結果為 0 就呼叫 `sk_free()`
  - [2] `sk_free()`
    - sock's wmem refcnt -= 1
    - 若結果為 0 就呼叫 `__sk_free()`

```c
static inline void sock_put(struct sock *sk)
{
    if (refcount_dec_and_test(&sk->sk_refcnt))
        sk_free(sk);
}
```

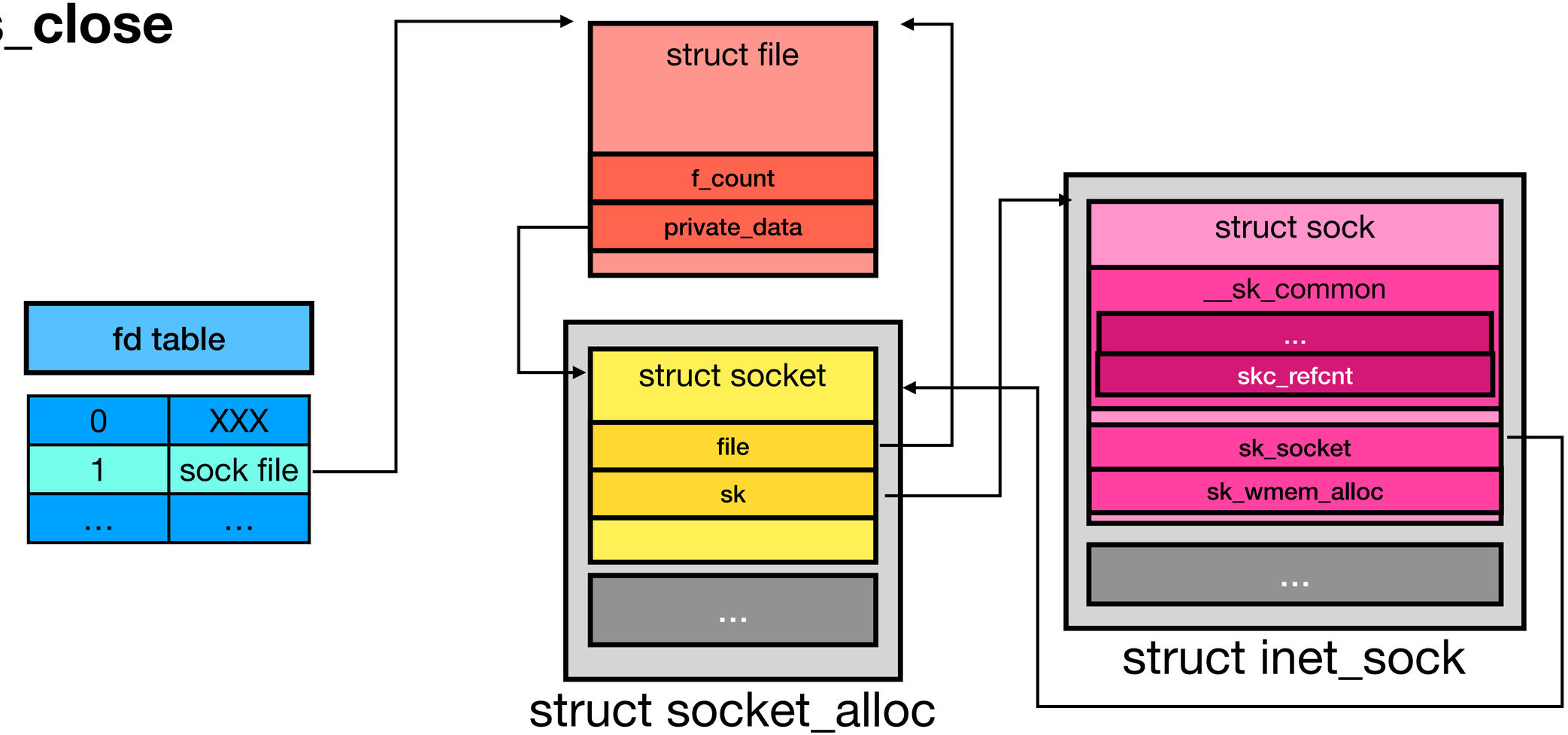
[1]

```c
void sk_free(struct sock *sk)
{
    if (refcount_dec_and_test(&sk->sk_wmem_alloc))
        __sk_free(sk);
}
```

[2]

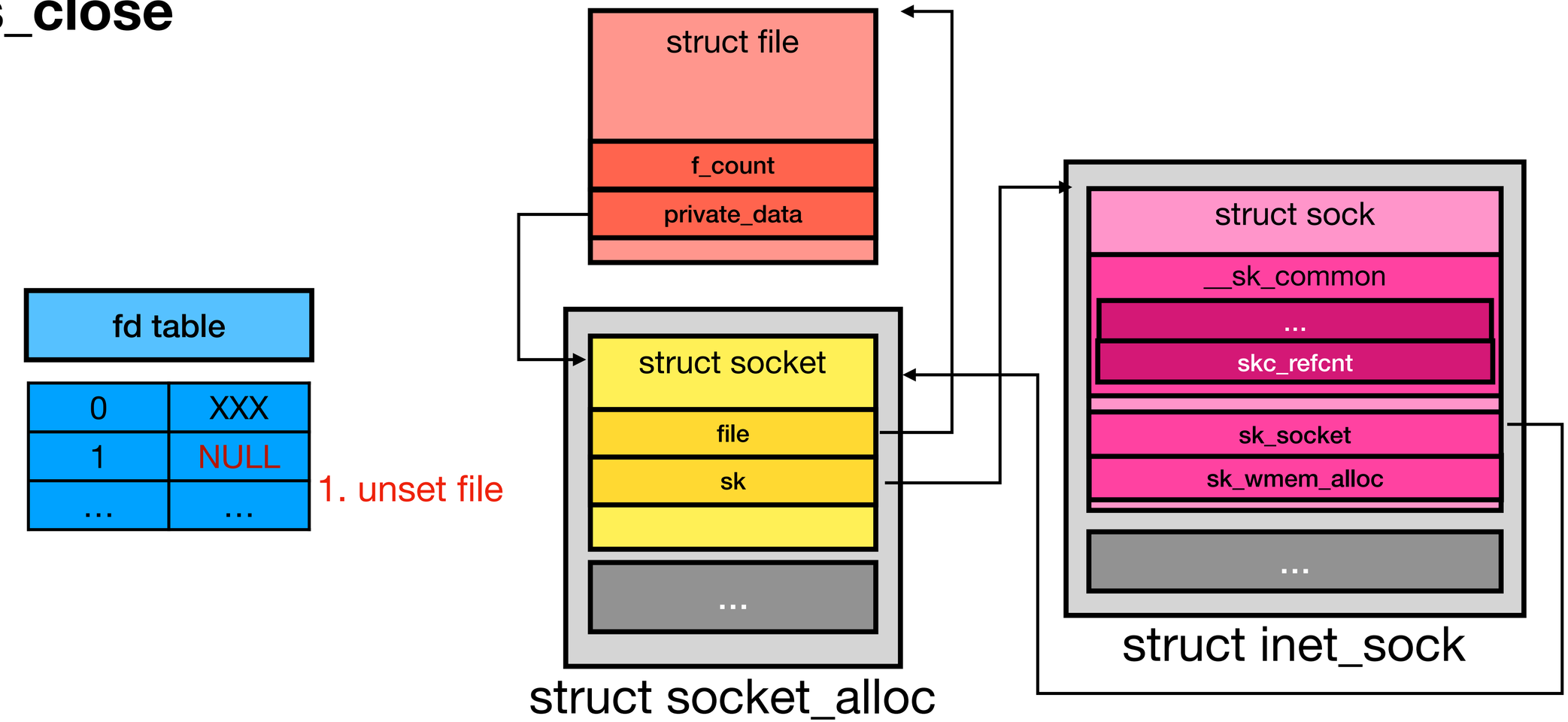
Overview

sys_close



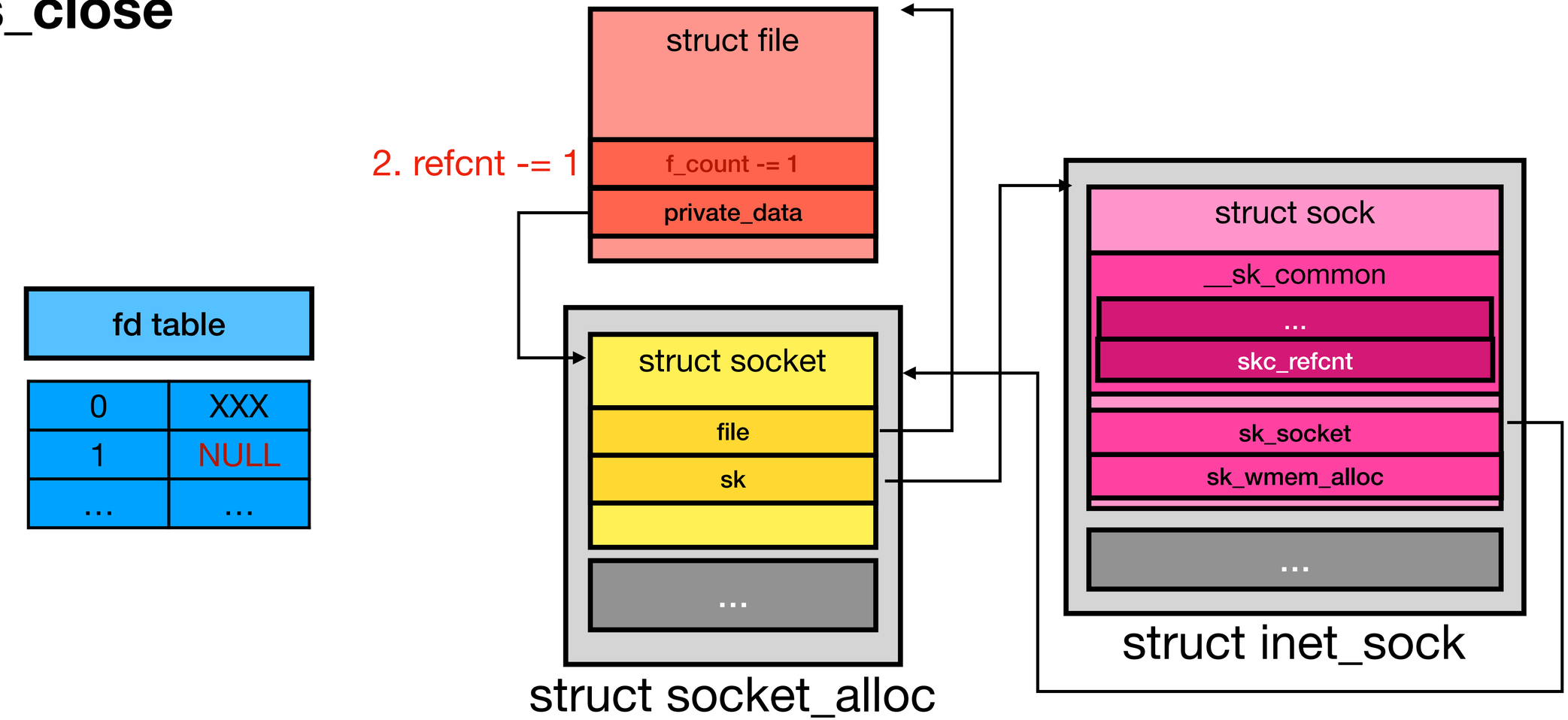
Overview

sys_close



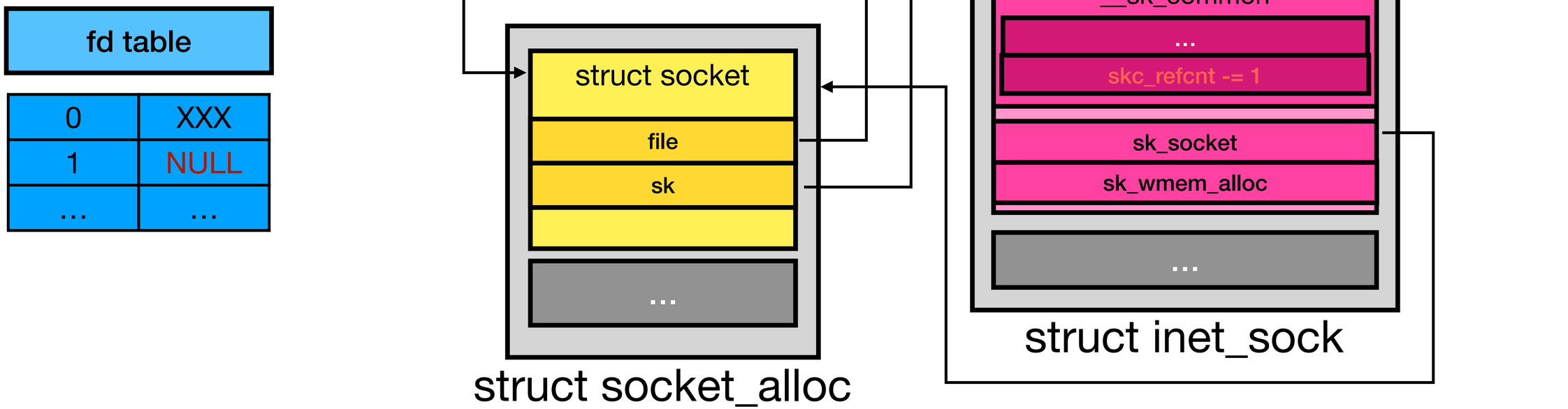
Overview

sys_close



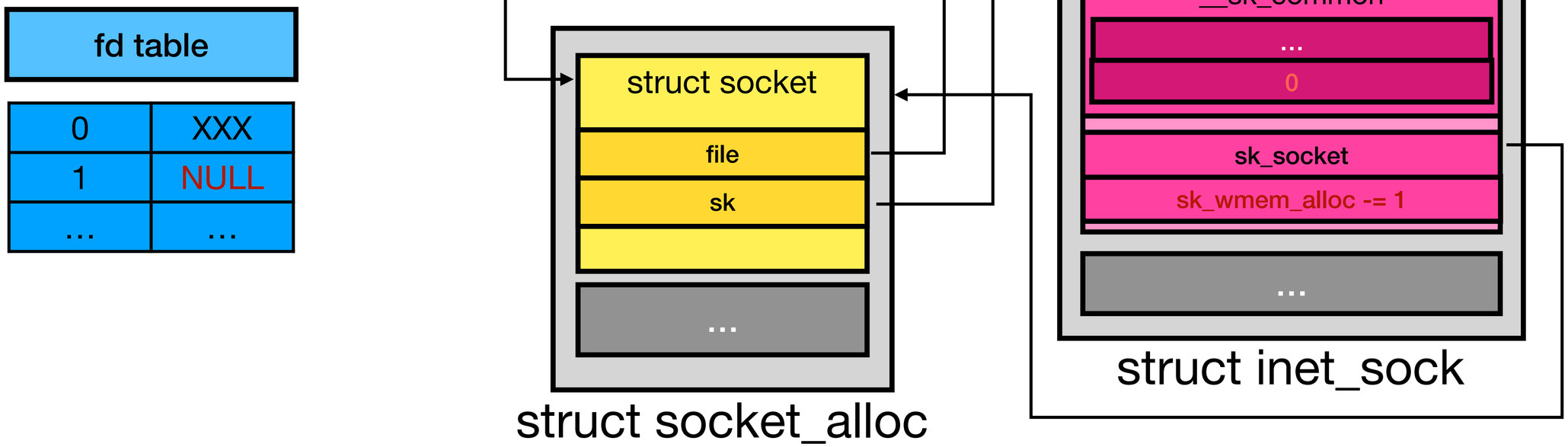
Overview

sys_close



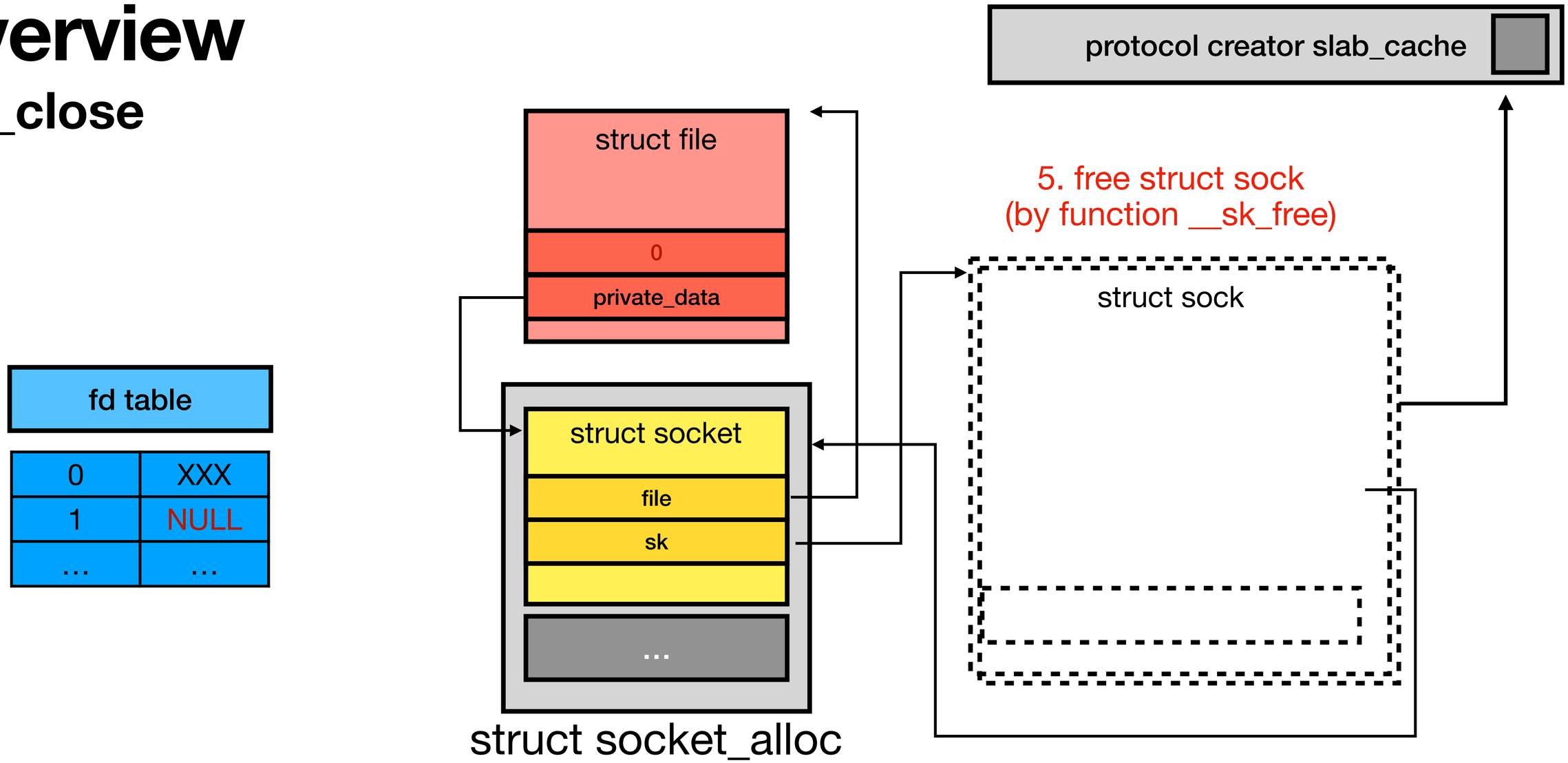
Overview

sys_close



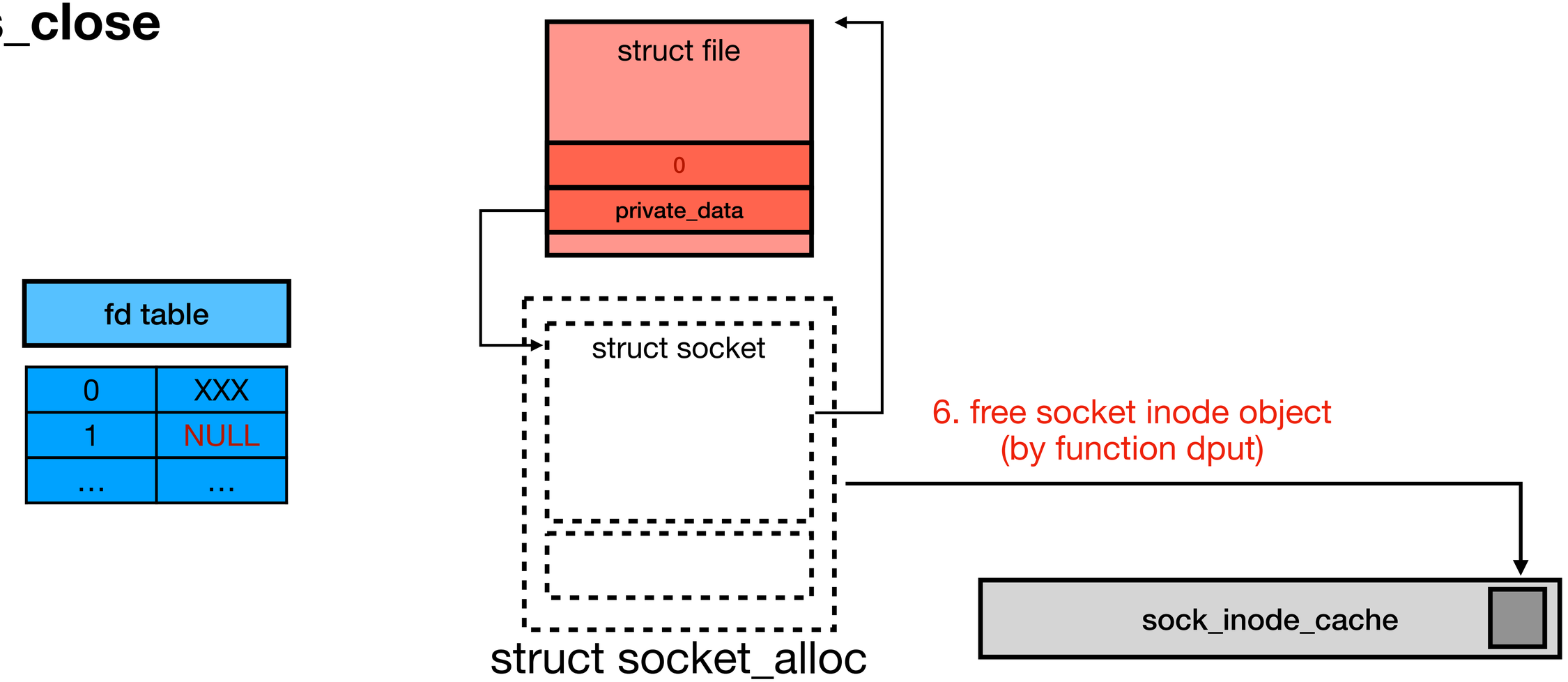
Overview

sys_close



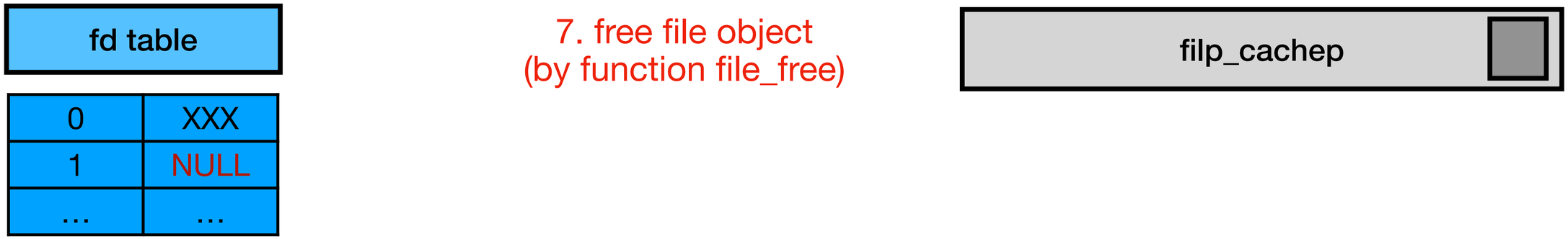
Overview

sys_close



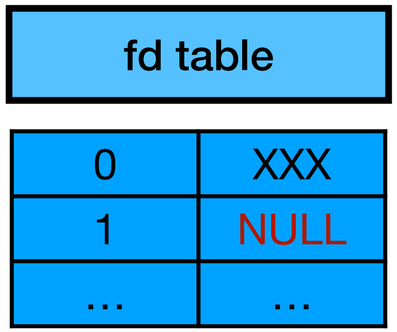
Overview

sys_close



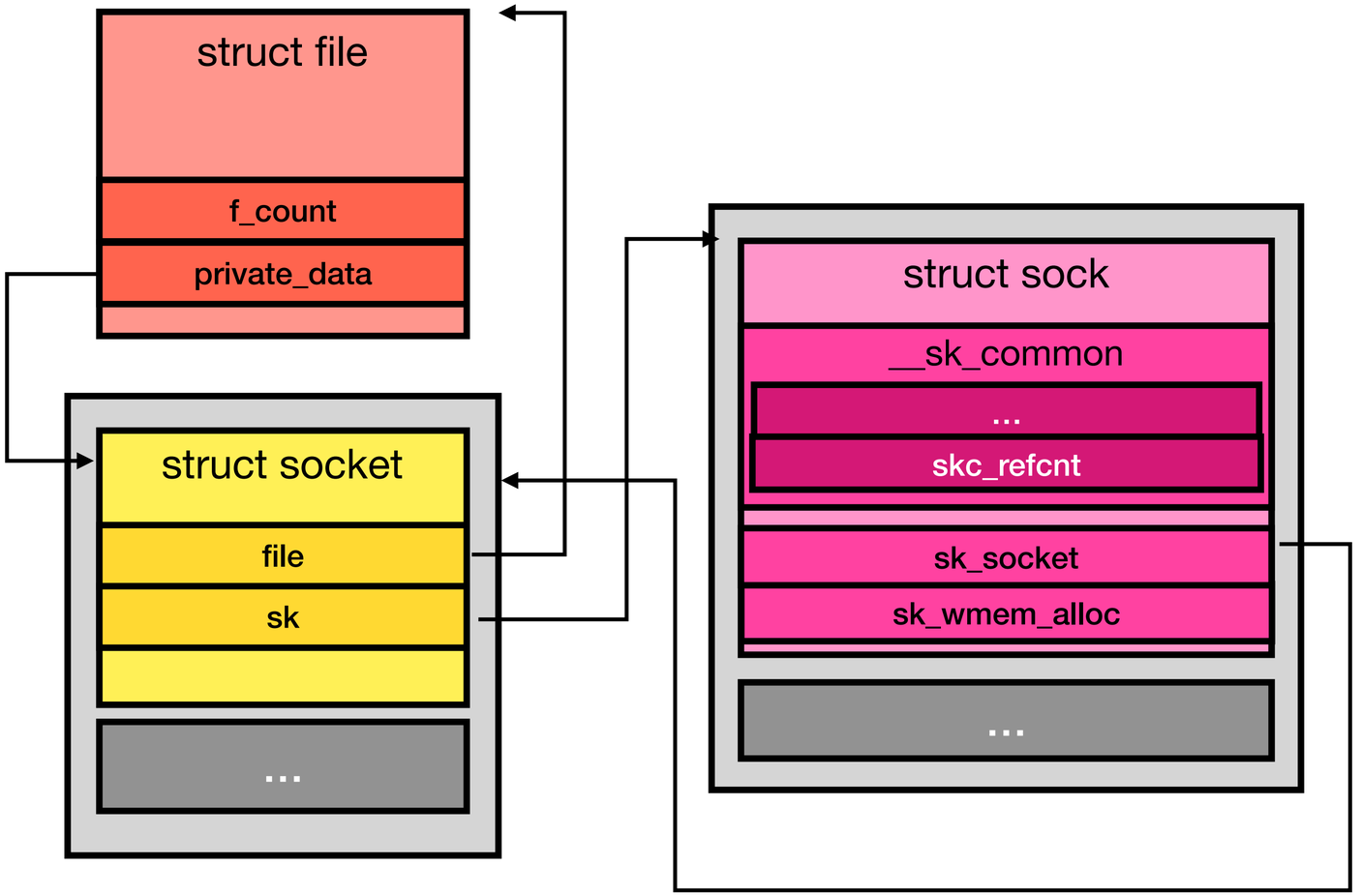
Overview

sys_close



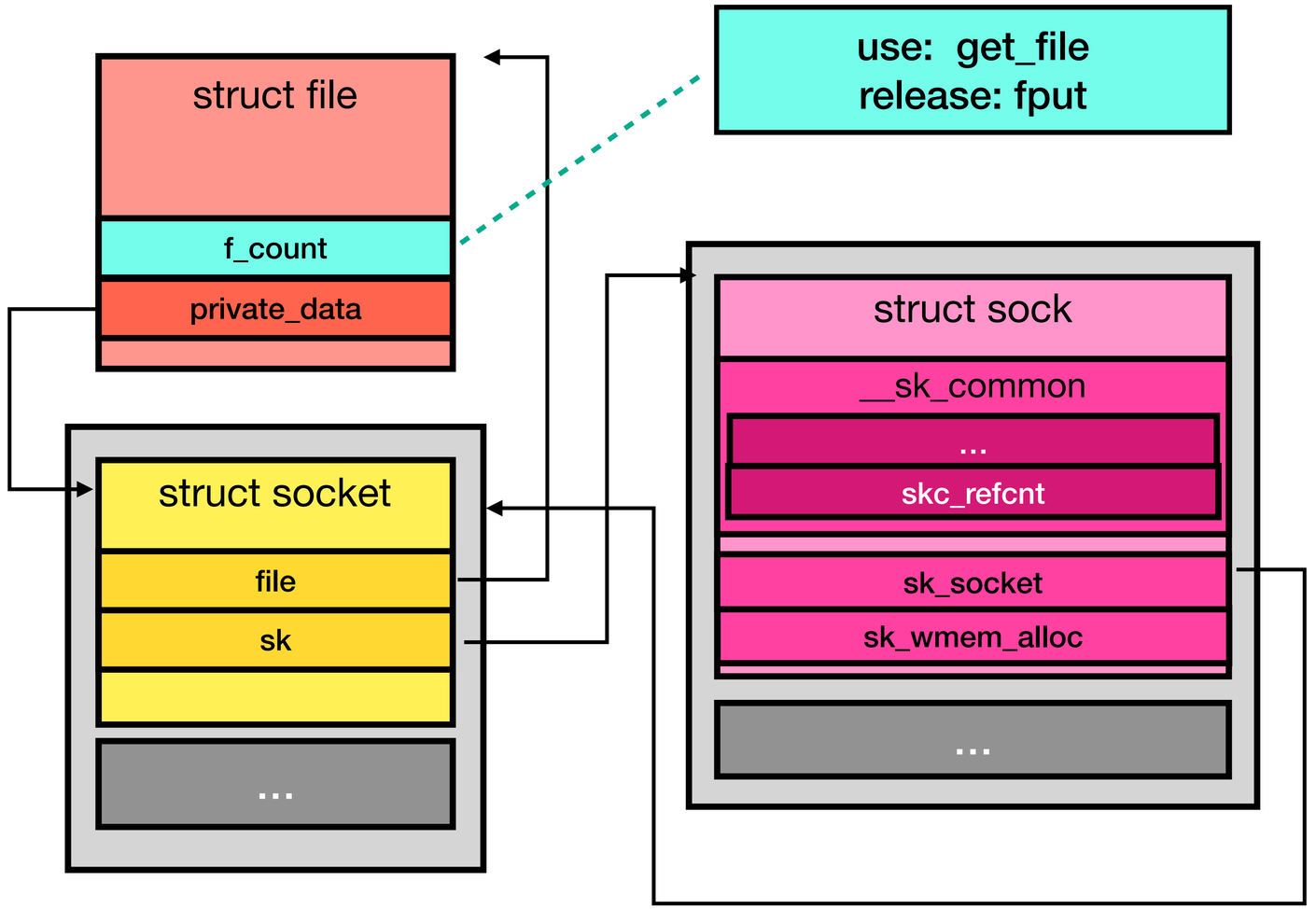
Overview

sys_close



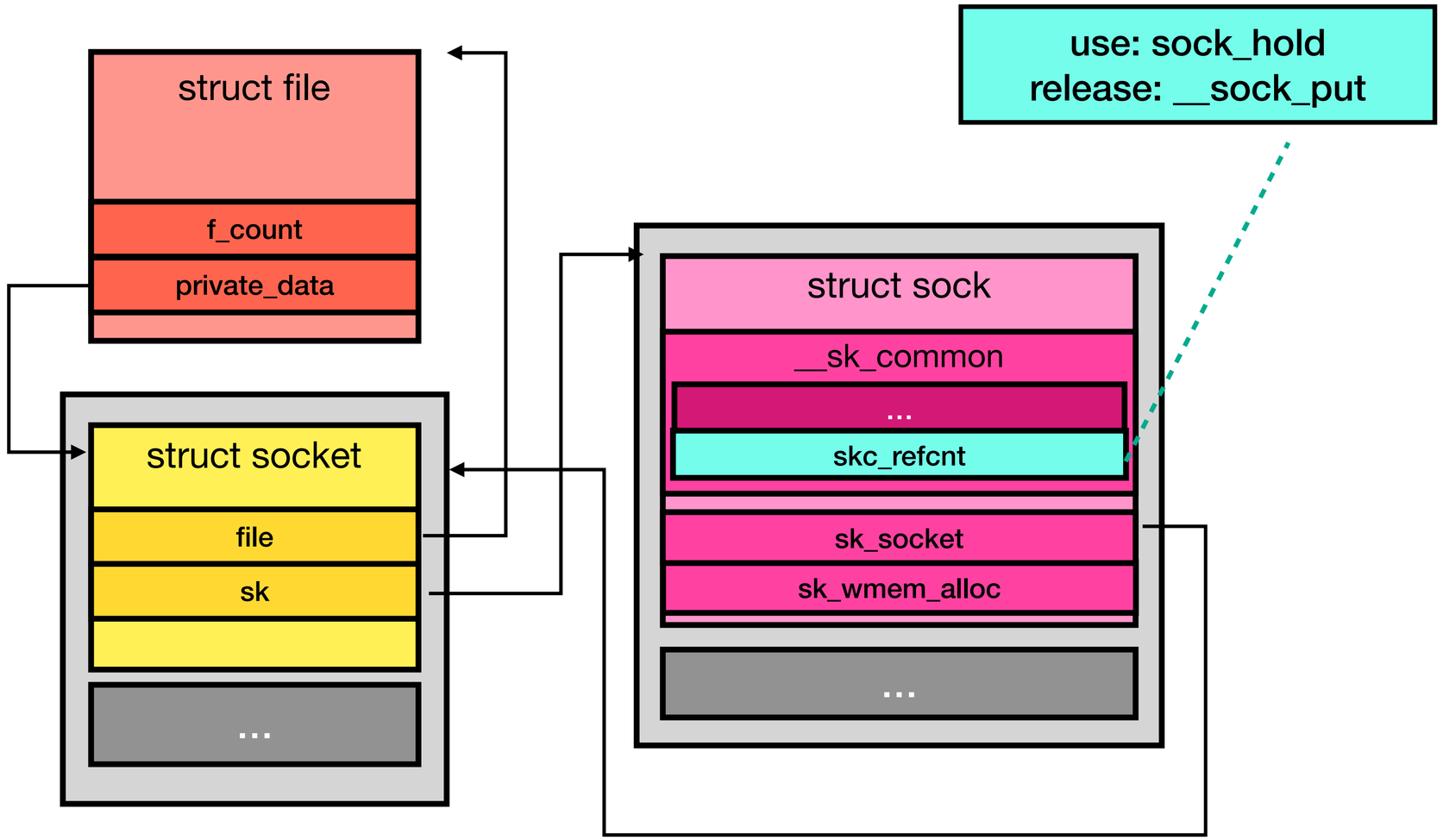
Overview

Summary



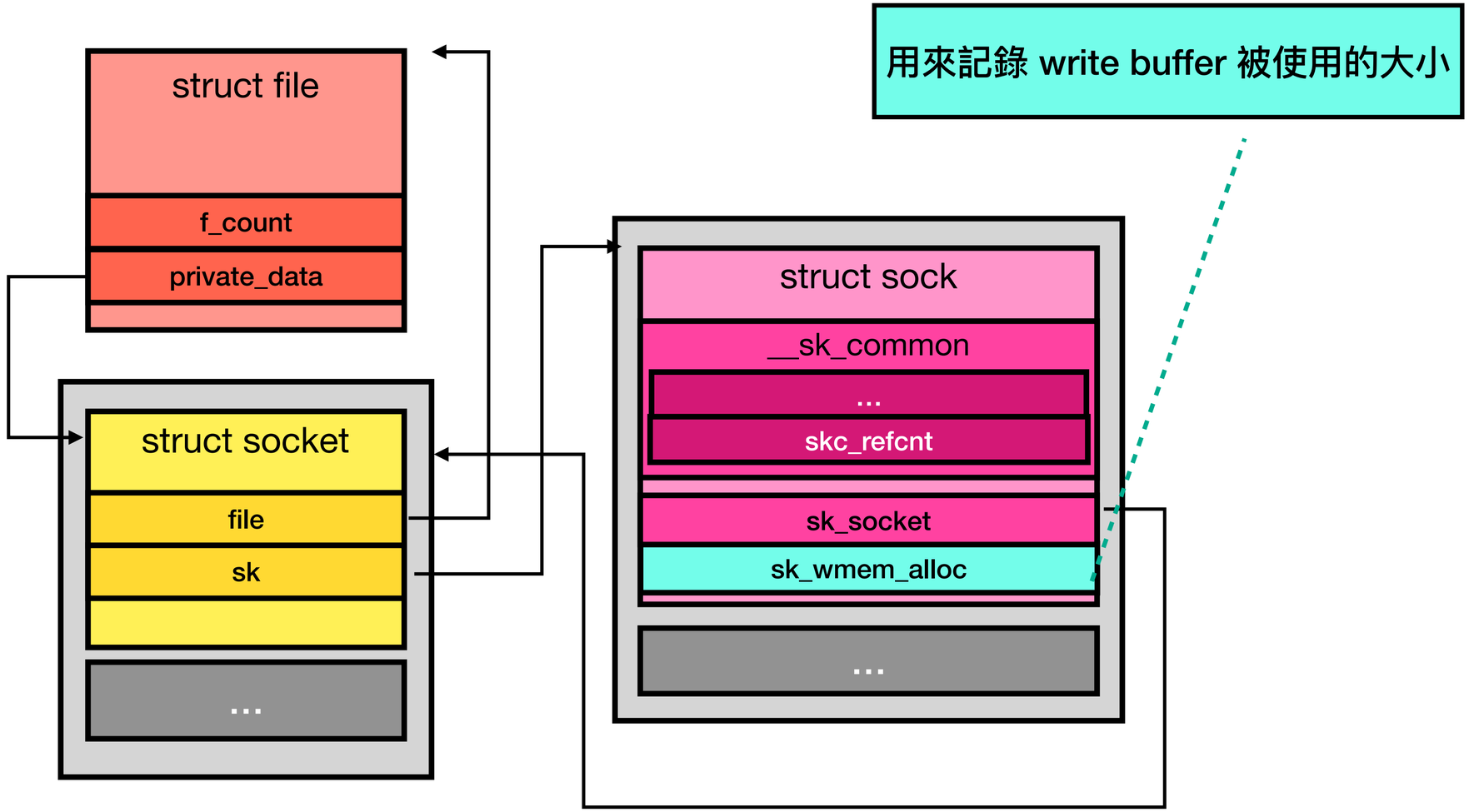
Overview

Summary



Overview

Summary



Overview

sys_{set, get}sockopt

- int getsockopt(int sockfd, int level, int optname, void optval[restrict *.optlen], socklen_t *restrict optlen);
- int setsockopt(int sockfd, int level, int optname, const void optval[.optlen], socklen_t optlen);

```
SYSCALL_DEFINE5(setsockopt, int, fd, int, level, int, optname,
        char __user *, optval, int, optlen)
{
    return __sys_setsockopt(fd, level, optname, optval, optlen);
}
```

```
SYSCALL_DEFINE5(getsockopt, int, fd, int, level, int, optname,
        char __user *, optval, int __user *, optlen)
{
    return __sys_getsockopt(fd, level, optname, optval, optlen);
}
```

Overview

sys_{set, get}sockopt

- Call `sockfd_lookup_light()`
- 若 Level 指定 SOL_SOCKET，代表操作的是 socket level 的設定，會有特別處理
  - `sock_{set, get}sockopt()`
- 否則 sock->ops->{set, get}sockopt
  - Call type {set, get}sockopt handler

```
ops = READ_ONCE(sock->ops);
if (level == SOL_SOCKET && !sock_use_custom_sol_socket(sock))
    err = sock_setsockopt(sock, level, optname, optval, optlen);
else if (unlikely(!ops->setsockopt))
    err = -EOPNOTSUPP;
else
    err = ops->setsockopt(sock, level, optname, optval,
                          optlen);
```

setsockopt

```
ops = READ_ONCE(sock->ops);
if (level == SOL_SOCKET)
    err = sock_getsockopt(sock, level, optname, optval, optlen);
else if (unlikely(!ops->getsockopt))
    err = -EOPNOTSUPP;
else
    err = ops->getsockopt(sock, level, optname, optval,
                          optlen);
```

getsockopt

Overview

sys_{set, get}sockopt

- Call `sockfd_lookup_light()`
- 若 Level 指定 SOL_SOCKET，代表操作的是 socket level 的設定，會有特別處理
  - `sock_{set, get}sockopt()`
- 否則 sock->ops->{set, get}sockopt
  - Call type {set, get}sockopt handler

```
ops = READ_ONCE(sock->ops);
if (level == SOL_SOCKET && !sock_use_custom_sol_socket(sock))
    err = sock_setsockopt(sock, level, optname, optval, optlen);
else if (unlikely(!ops->setsockopt))
    err = -EOPNOTSUPP;
else
    err = ops->setsockopt(sock, level, optname, optval,
                          optlen);
```

setsockopt

```
ops = READ_ONCE(sock->ops);
if (level == SOL_SOCKET)
    err = sock_getsockopt(sock, level, optname, optval, optlen);
else if (unlikely(!ops->getsockopt))
    err = -EOPNOTSUPP;
else
    err = ops->getsockopt(sock, level, optname, optval,
                          optlen);
```

getsockopt

Overview

sys_{set, get}sockopt

- `sock_{set, get}sockopt()`
  - 最後走到 `sk_setsockopt()`
  - 根據不同的 option name 回傳/設置對應的 option
    - SO_SNDBUF
    - SO_RCVBUF
    - …

```
int sk_setsockopt(struct sock *sk, int level, int optname,
                  sockptr_t optval, unsigned int optlen)
{
    // [...]

    switch (optname) {
    case SO_DEBUG:
        if (val && !sockopt_capable(CAP_NET_ADMIN))
            ret = -EACCES;
        else
            sock_valbool_flag(sk, SOCK_DBG, valbool);
        break;
    case SO_REUSEADDR:
        sk->sk_reuse = (valbool ? SK_CAN_REUSE : SK_NO_REUSE);
        break;
    case SO_REUSEPORT:
        sk->sk_reuseport = valbool;
        break;
    case SO_TYPE:
    case SO_PROTOCOL:
    case SO_DOMAIN:
    case SO_ERROR:
        ret = -ENOPROTOOPT;
        break;
```

Overview

sys_{set, get}sockopt

- Type {set, get}sockopt handler
  - Call protocol socket {set, get}sockopt handler
- Protocol recvmsg handler
  - 若 level 為 SOL_TCP，
    呼叫 `do_tcp_{set, get}sockopt()`
  - 否則呼叫 `icsk->icsk_af_ops->{set, get}sockopt()`
    - 設置 IP level 的 option

```
const struct inet_connection_sock_af_ops ipv4_specific = {
    .queue_xmit     = ip_queue_xmit,
    .send_check     = tcp_v4_send_check,
    .rebuild_header     = inet_sk_rebuild_header,
    .sk_rx_dst_set      = inet_sk_rx_dst_set,
    .conn_request       = tcp_v4_conn_request,
    .syn_recv_sock      = tcp_v4_syn_recv_sock,
    .net_header_len     = sizeof(struct iphdr),
    .setsockopt     = ip_setsockopt,
    .getsockopt     = ip_getsockopt,
    .addr2sockaddr      = inet_csk_addr2sockaddr,
    .sockaddr_len       = sizeof(struct sockaddr_in),
    .mtu_reduced        = tcp_v4_mtu_reduced,
};
```

Overview

sys_{set, get}sockopt

- Type {set, get}sockopt handler
  - Call protocol socket {set, get}sockopt handler
- Protocol recvmsg handler
  - 若 level 為 SOL_TCP，
    呼叫 `do_tcp_{set, get}sockopt()`
  - 否則呼叫 `icsk->icsk_af_ops->{set, get}sockopt()`
    - 設置 IP level 的 option

```
const struct inet_connection_sock_af_ops ipv4_specific = {
    .queue_xmit     = ip_queue_xmit,
    .send_check     = tcp_v4_send_check,
    .rebuild_header     = inet_sk_rebuild_header,
    .sk_rx_dst_set      = inet_sk_rx_dst_set,
    .conn_request       = tcp_v4_conn_request,
    .syn_recv_sock      = tcp_v4_syn_recv_sock,
    .net_header_len     = sizeof(struct iphdr),
    .setsockopt     = ip_setsockopt,
    .getsockopt     = ip_getsockopt,
    .addr2sockaddr      = inet_csk_addr2sockaddr,
    .sockaddr_len       = sizeof(struct sockaddr_in),
    .mtu_reduced        = tcp_v4_mtu_reduced,
};
```

Overview

sys_{set, get}sockopt

- `do_tcp_{set, get}sockopt()`
  - TCP_ULP - 啟用 TLS, MPTCP, …
  - TCP_CORK - 可以 queue non-full frame
  - …

```
switch (optname) {
case TCP_CONGESTION: {…
case TCP_ULP: {…
case TCP_FASTOPEN_KEY: {…
default:
    /* fallthru */
    break;
}

if (optlen < sizeof(int))
    return -EINVAL;

if (copy_from_sockptr(&val, optval, sizeof(val))
    return -EFAULT;

/* Handle options that can be set without lockir
switch (optname) {
case TCP_SYNCNT:
    return tcp_sock_set_syncnt(sk, val);
case TCP_USER_TIMEOUT:
    return tcp_sock_set_user_timeout(sk, val);
case TCP_KEEPINTVL:
    return tcp_sock_set_keepintvl(sk, val);
case TCP_KEEPCNT:
    return tcp_sock_set_keepcnt(sk, val);
```

Overview

sys_{set, get}sockopt

- Type {set, get}sockopt handler
  - Call protocol socket {set, get}sockopt handler
- Protocol recvmsg handler
  - 若 level 為 SOL_TCP，
    呼叫 `do_tcp_{set, get}sockopt()`
  - 否則呼叫 `icsk->icsk_af_ops->{set, get}sockopt()`
    - 設置 IP level 的 option

```
const struct inet_connection_sock_af_ops ipv4_specific = {
    .queue_xmit     = ip_queue_xmit,
    .send_check     = tcp_v4_send_check,
    .rebuild_header     = inet_sk_rebuild_header,
    .sk_rx_dst_set      = inet_sk_rx_dst_set,
    .conn_request       = tcp_v4_conn_request,
    .syn_recv_sock      = tcp_v4_syn_recv_sock,
    .net_header_len     = sizeof(struct iphdr),
    .setsockopt     = ip_setsockopt,
    .getsockopt     = ip_getsockopt,
    .addr2sockaddr      = inet_csk_addr2sockaddr,
    .sockaddr_len       = sizeof(struct sockaddr_in),
    .mtu_reduced        = tcp_v4_mtu_reduced,
};
```

Overview

sys_{set, get}sockopt

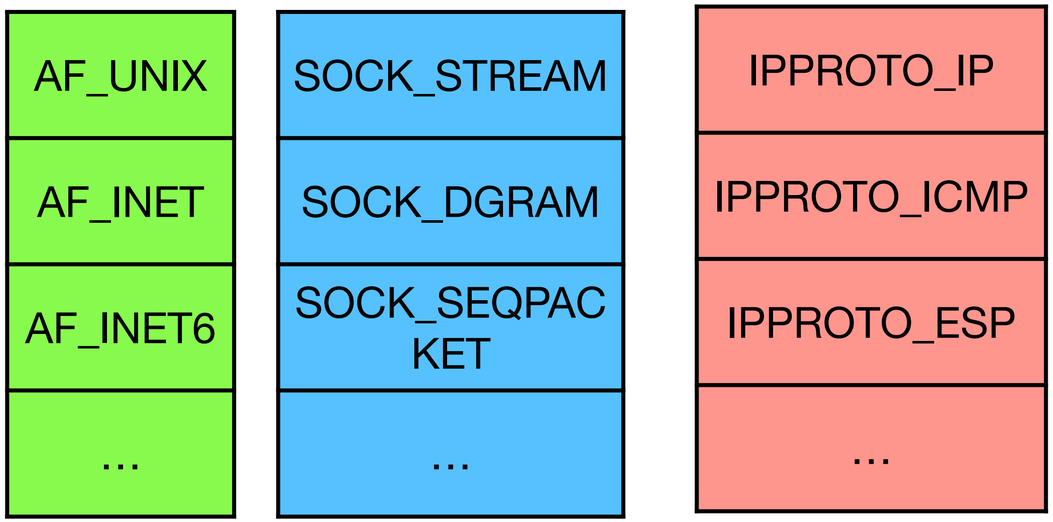
- IP connection socket {set, get}sockopt handler (以 ipv4_specific 為例)
  - Call do_ip_{set, get}sockopt
  - 能有一些奇奇怪怪的設定

```
switch (optname) {
case IP_PKTINFO:
    inet_assign_bit(PKTINFO, sk, val);
    return 0;
case IP_RECVTTL:
    inet_assign_bit(TTL, sk, val);
    return 0;
case IP_RECVTOS:
    inet_assign_bit(TOS, sk, val);
    return 0;
case IP_RECVOPTS:
    inet_assign_bit(RECVOPTS, sk, val);
    return 0;
case IP_RETOPTS:
    inet_assign_bit(RETOPTS, sk, val);
    return 0;
case IP_PASSSEC:
    inet_assign_bit(PASSSEC, sk, val);
    return 0;
case IP_RECVORIGDSTADDR:
    inet_assign_bit(ORIGDSTADDR, sk, val);
    return 0;
```

Overview

Summary

socket(family, type, protocol)



Overview

Summary

socket(family, type, protocol)

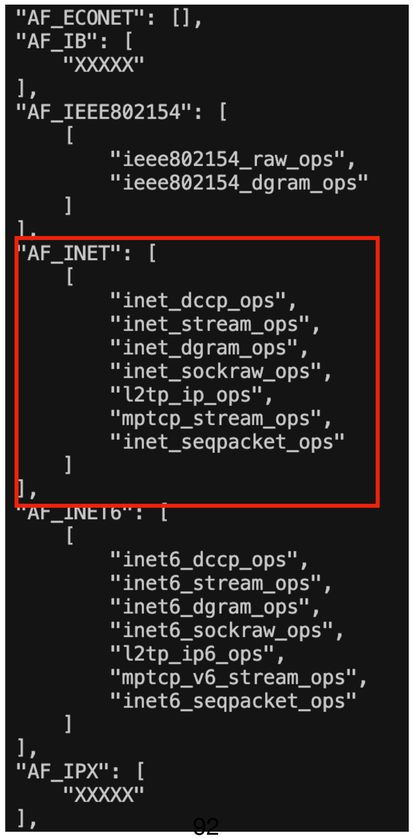
```
#define AF_UNSPEC   0
#define AF_UNIX     1   /* Unix domain sockets      */
#define AF_LOCAL    1   /* POSIX name for AF_UNIX   */
#define AF_INET     2   /* Internet IP Protocol     */
#define AF_AX25     3   /* Amateur Radio AX.25      */
#define AF_IPX      4   /* Novell IPX           */
#define AF_APPLETALK    5   /* AppleTalk DDP        */
#define AF_NETROM   6   /* Amateur Radio NET/ROM    */
#define AF_BRIDGE   7   /* Multiprotocol bridge     */
#define AF_ATMPVC   8   /* ATM PVCs         */
#define AF_X25      9   /* Reserved for X.25 project    */
#define AF_INET6    10  /* IP version 6         */
#define AF_ROSE     11  /* Amateur Radio X.25 PLP   */
#define AF_DECnet   12  /* Reserved for DECnet project  */
#define AF_NETBEUI  13  /* Reserved for 802.2LLC project*/
#define AF_SECURITY 14  /* Security callback pseudo AF */
#define AF_KEY      15      /* PF_KEY key management API */
#define AF_NETLINK  16
// [...]
```

一共 46 種

Overview

Summary

socket(family, type, protocol)



每個 type 在不同 family 有自己的實作，共有 3X 個

Overview

Summary

socket(family, type, protocol)

UDP

ICMP

IP

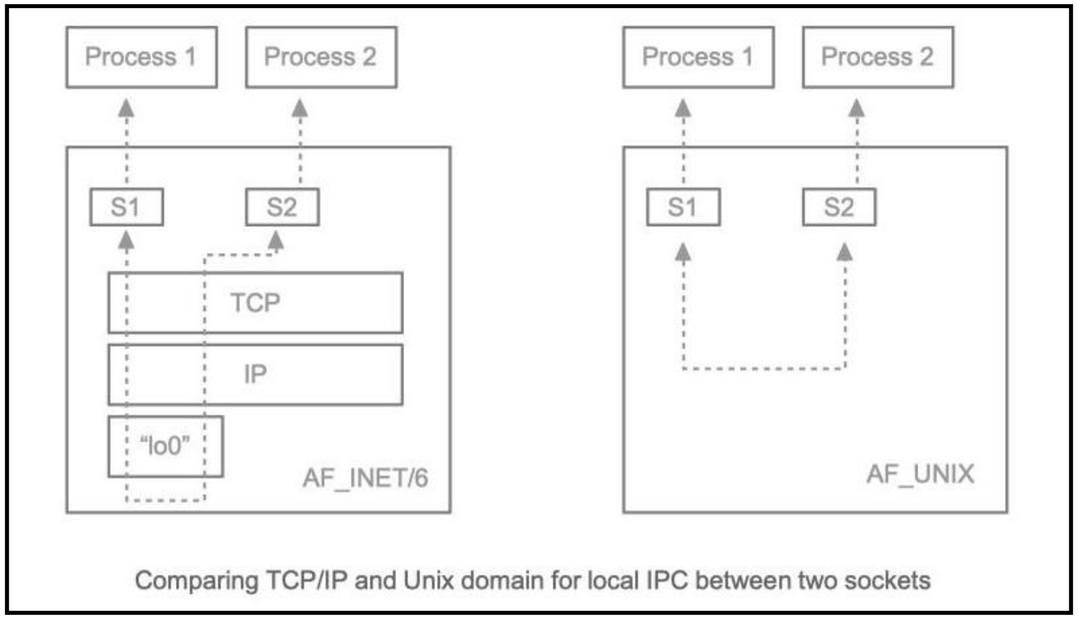
```
{
    .type =       SOCK_DGRAM,
    .protocol =   IPPROTO_UDP,
    .prot =       &udp_prot,
    .ops =        &inet_dgram_ops,
    .flags =      INET_PROTOSW_PERMANENT,
},

{
    .type =       SOCK_DGRAM,
    .protocol =   IPPROTO_ICMP,
    .prot =       &ping_prot,
    .ops =        &inet_sockraw_ops,
    .flags =      INET_PROTOSW_REUSE,
},

{
    .type =       SOCK_RAW,
    .protocol =   IPPROTO_IP,   /* wild ca
    .prot =       &raw_prot,
    .ops =        &inet_sockraw_ops,
    .flags =      INET_PROTOSW_REUSE,
}
```

同個 type 可能會有不同 protocol

AF_UNIX

AF_UNIX
Overview

- 一種 Linux kernel 的 IPC 機制



Comparing TCP/IP and Unix domain for local IPC between two sockets

https://inside.java/2021/02/03/jep380-unix-domain-sockets-channels/

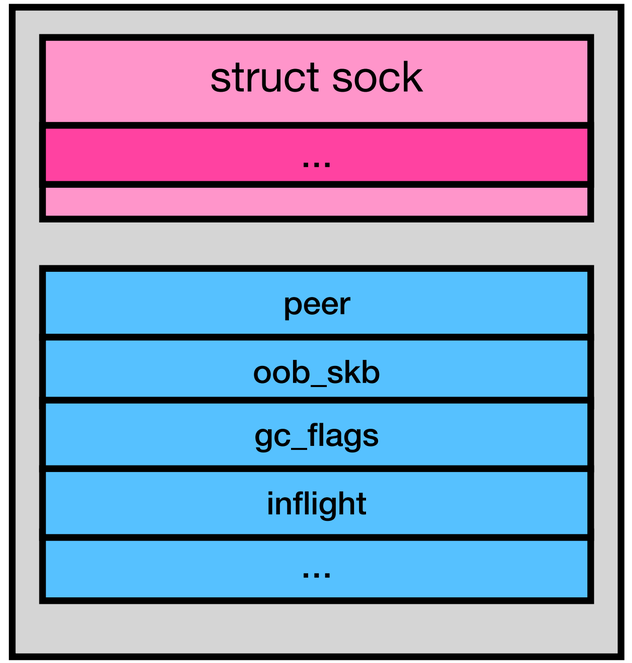
AF_UNIX
Overview

- 共有三種 proto_ops (type)
  - unix_stream_ops
  - unix_dgram_ops
  - unix_seqpacket_ops

```
switch (sock->type) {
case SOCK_STREAM:
    sock->ops = &unix_stream_ops;
    break;
case SOCK_RAW:
    sock->type = SOCK_DGRAM;
    fallthrough;
case SOCK_DGRAM:
    sock->ops = &unix_dgram_ops;
    break;
case SOCK_SEQPACKET:
    sock->ops = &unix_seqpacket_ops;
    break;
default:
    return -ESOCKTNOSUPPORT;
}
```

AF_UNIX
Overview

- unix_create - create + init sock object
  - Create - `sk_alloc(PF_UNIX, …)`
  - Init - `sock_init_data(sock, sk)`
  - 不同 family 的 sock object 會有自己的結構
    - AF_UNIX - struct unix_sock
    - AF_INET - struct inet_sock
    - …



struct unix_sock

AF_UNIX
Overview

- 共有三種 proto_ops (type)
  - unix_stream_ops
  - unix_dgram_ops
  - unix_seqpacket_ops

```
switch (sock->type) {
case SOCK_STREAM:
    sock->ops = &unix_stream_ops;
    break;
case SOCK_RAW:
    sock->type = SOCK_DGRAM;
    fallthrough;
case SOCK_DGRAM:
    sock->ops = &unix_dgram_ops;
    break;
case SOCK_SEQPACKET:
    sock->ops = &unix_seqpacket_ops;
    break;
default:
    return -ESOCKTNOSUPPORT;
}
```

AF_UNIX

unix_stream_ops

- [1] 都是由 type ops (proto_ops) 完成請求
- [2] protocol ops 沒做事情

```
static const struct proto_ops unix_stream_ops = {
    .family =   PF_UNIX,
    .owner =    THIS_MODULE,
    .release =  unix_release,
    .bind =     unix_bind,
    .connect =  unix_stream_connect,
    .socketpair =   unix_socketpair,
    .accept =   unix_accept,
    .getname =  unix_getname,
    .poll =     unix_poll,
    .ioctl =    unix_ioctl,
    // [...]
```

[1]

```
struct proto unix_stream_proto = {
    .name           = "UNIX-STREAM",
    .owner          = THIS_MODULE,
    .obj_size       = sizeof(struct unix_sock),
    .close          = unix_close,
    .unhash         = unix_unhash,
    .bpf_bypass_getsockopt  = unix_bpf_bypass_getsockopt,
#ifdef CONFIG_BPF_SYSCALL
    .psock_update_sk_prot   = unix_stream_bpf_update_proto,
#endif
};
```

[2]

AF_UNIX

unix_stream_ops

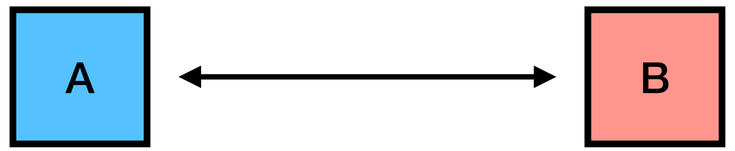
- socketpair (unix_socketpair)
- sendmsg (unix_stream_sendmsg)
- recvmsg (unix_stream_recvmsg)
- close (unix_release)

AF_UNIX

unix_stream_ops

- socketpair (unix_socketpair)
  - 建立一組 sock A 跟 B 互相連接
  - Socket state = CONNECTED
  - Sock state = ESTABLISHED
  - 彼此的 sock refcnt 都 +=1
- 其實就是雙向的 pipe

```c
static int unix_socketpair(struct socket *socka, struct socket *sockb)
{
    struct sock *ska = socka->sk, *skb = sockb->sk;

    /* Join our sockets back to back */
    sock_hold(ska);
    sock_hold(skb);
    unix_peer(ska) = skb;
    unix_peer(skb) = ska;
    init_peercred(ska);
    init_peercred(skb);

    ska->sk_state = TCP_ESTABLISHED;
    skb->sk_state = TCP_ESTABLISHED;
    socka->state  = SS_CONNECTED;
    sockb->state  = SS_CONNECTED;
    return 0;
}
```



AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [1] Wait gc
  - [2] Setup scm (Socket Control Message)
  - [3] Allocate packet skb
  - [4] Attach files in the scm to skb
  - …

AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [1] Wait gc
    - GC 在每次有 unix socket 被釋放或 inflight fd 超過 threshold 時觸發
    - GC 發生時 gc_in_progress = 1，syscall 會等 gc 結束才繼續執行

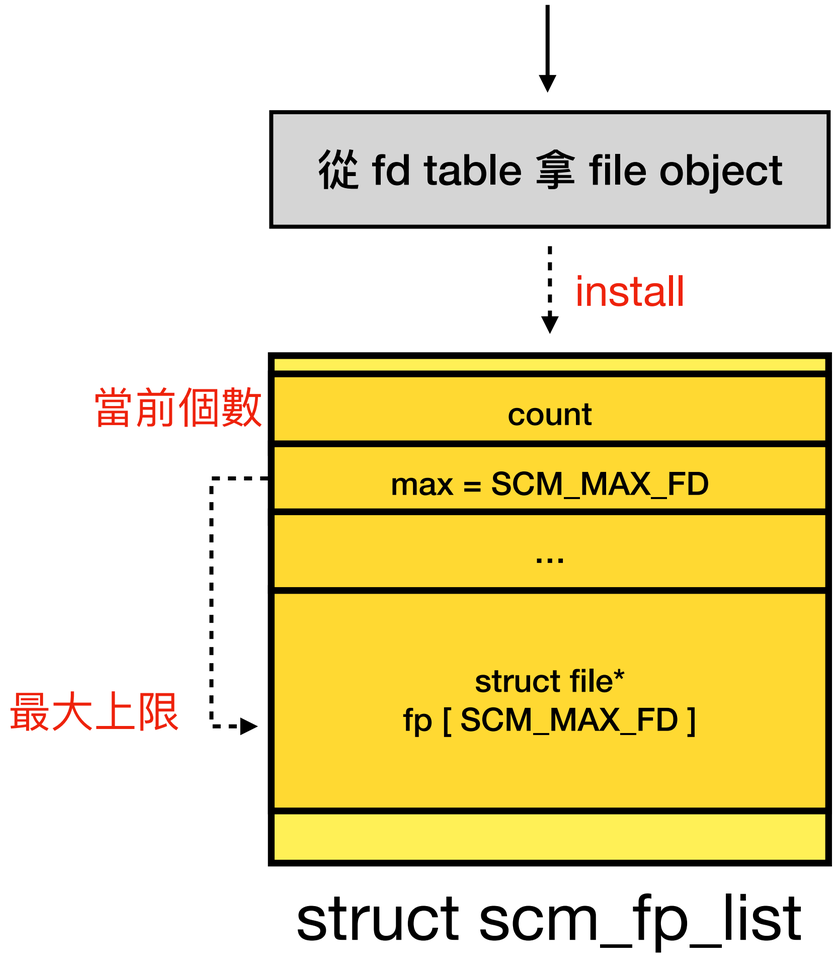
```
void wait_for_unix_gc(void)
{
    /* If number of inflight sockets is insane,
     * force a garbage collect right now.
     * Paired with the WRITE_ONCE() in unix_inflight(),
     * unix_notinflight() and gc_in_progress().
     */
    if (READ_ONCE(unix_tot_inflight) > UNIX_INFLIGHT_TRIGGER_GC &&
        !READ_ONCE(gc_in_progress))
        unix_gc();
    wait_event(unix_gc_wait, !READ_ONCE(gc_in_progress));
}
```

AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [2] Setup scm (Socket Control Message)
    - 使用者能透過 control msg 傳 file object 給 peer process
    - 一次請求能包含多個 cmsg，但總共只能塞 253 (SCM_MAX_FD) 個 fd

```
switch (cmsg->cmsg_type)
{
case SCM_RIGHTS:
    if (!ops || ops->family != PF_UNIX)
        goto error;
    err = scm_fp_copy(cmsg, &p->fp);
    if (err<0)
        goto error;
    break;
```



AF_UNIX

unix_stream_ops

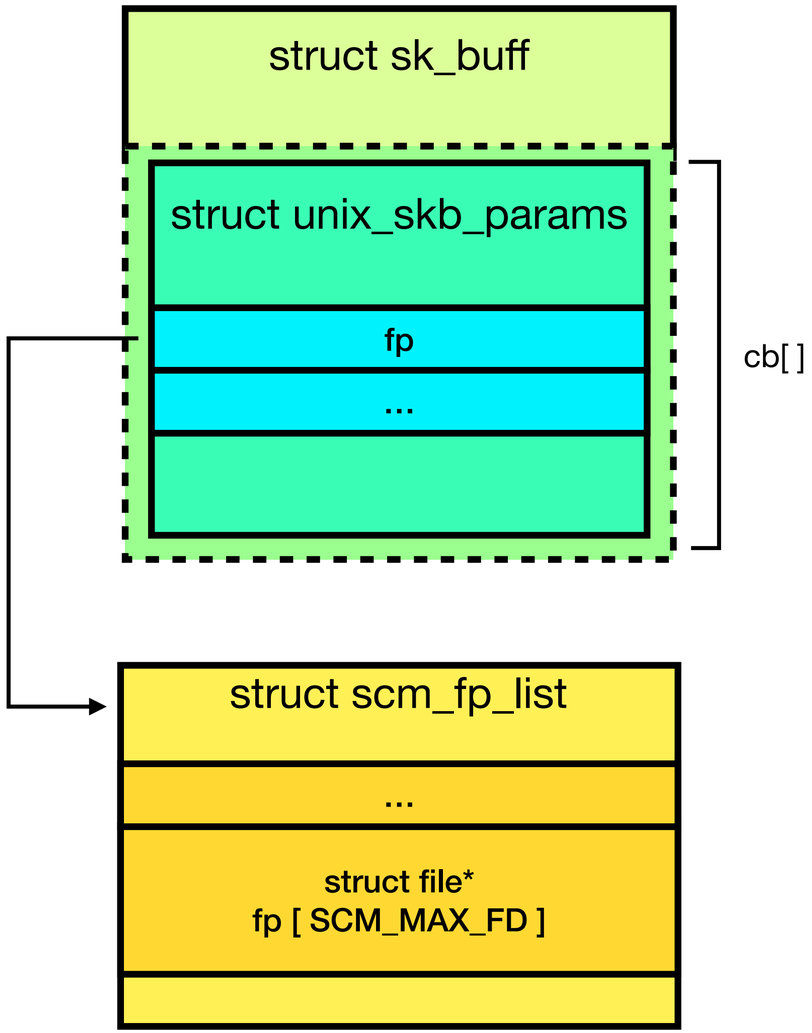
- sendmsg (unix_stream_sendmsg)
  - [3] Allocate packet skb
    - Packet size 不能超過 sndbuf size
    - 計算出 header size 與 data size 後分配 sk_buff
      - 在 data < PAGE_SIZE 時不會額外分配 fragment

```
size = min_t(int, size, (sk->sk_sndbuf >> 1) - 64);
size = min_t(int, size, SKB_MAX_HEAD(0) + UNIX_SKB_FRAGS_SZ);

data_len = max_t(int, 0, size - SKB_MAX_HEAD(0));
data_len = min_t(size_t, size, PAGE_ALIGN(data_len));

skb = sock_alloc_send_pskb(sk, size - data_len, data_len,
                           msg->msg_flags & MSG_DONTWAIT, &err,
                           get_order(UNIX_SKB_FRAGS_SZ));
```

AF_UNIX

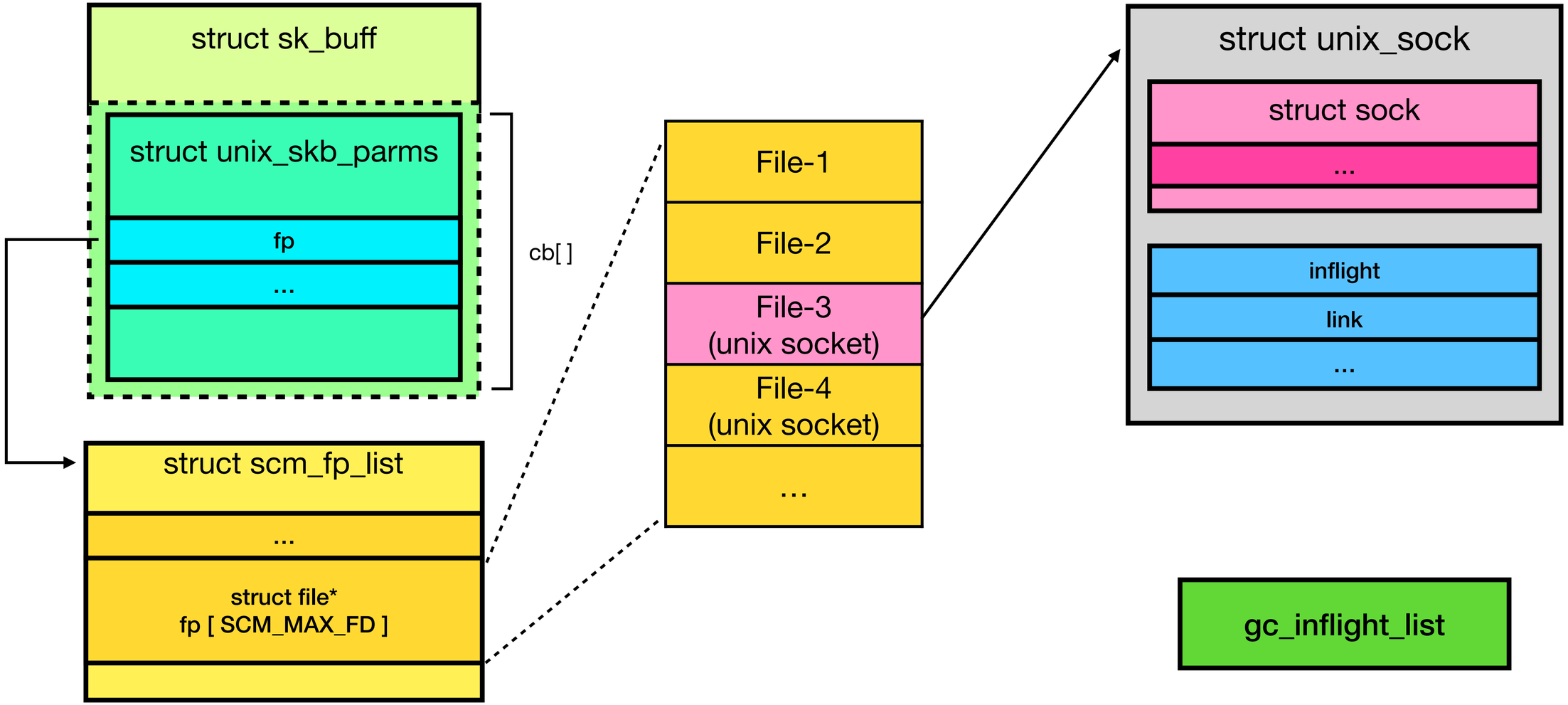
unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [4] Attach files in the scm to skb
    - dup 一份 scm object
    - Attack scm 內的 file to unix skb object
      - 若 file 為 unix_socket，則會被視為 inflight 並額外處理



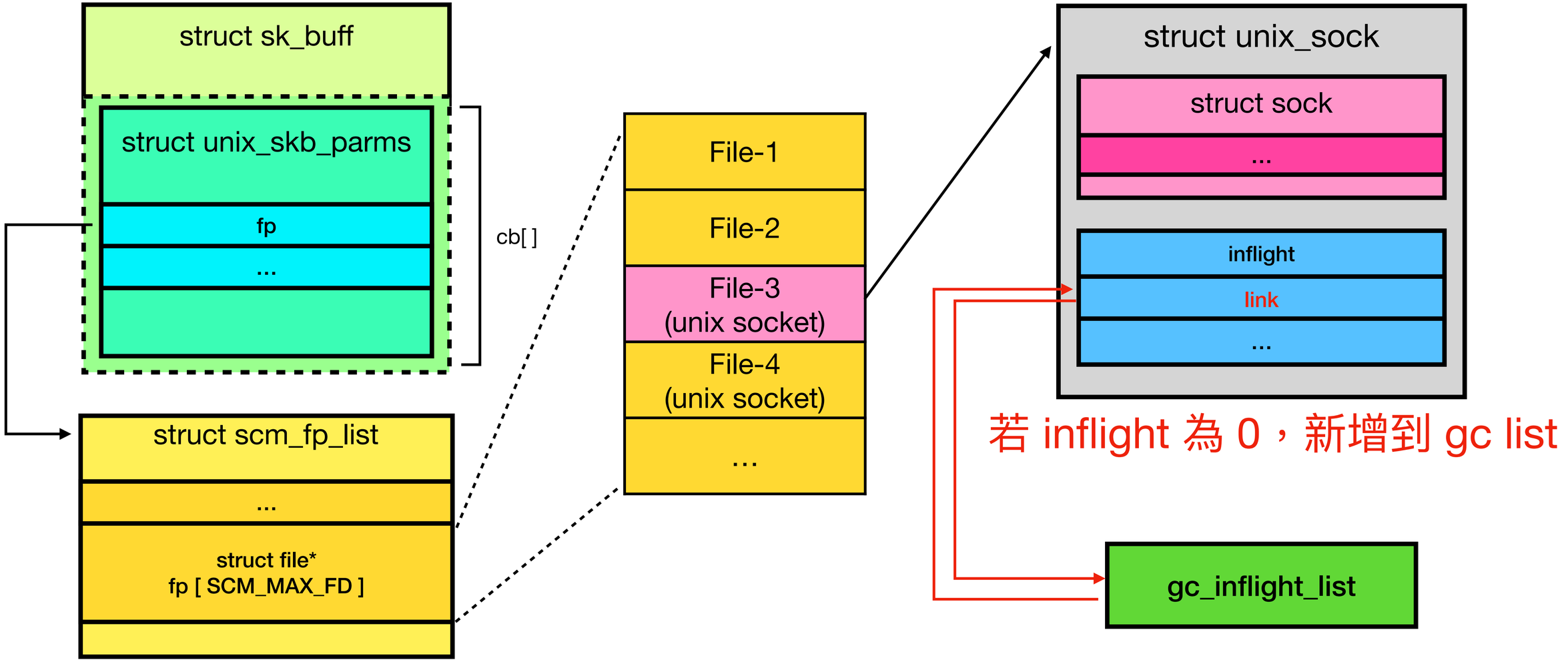
AF_UNIX

unix_stream_ops



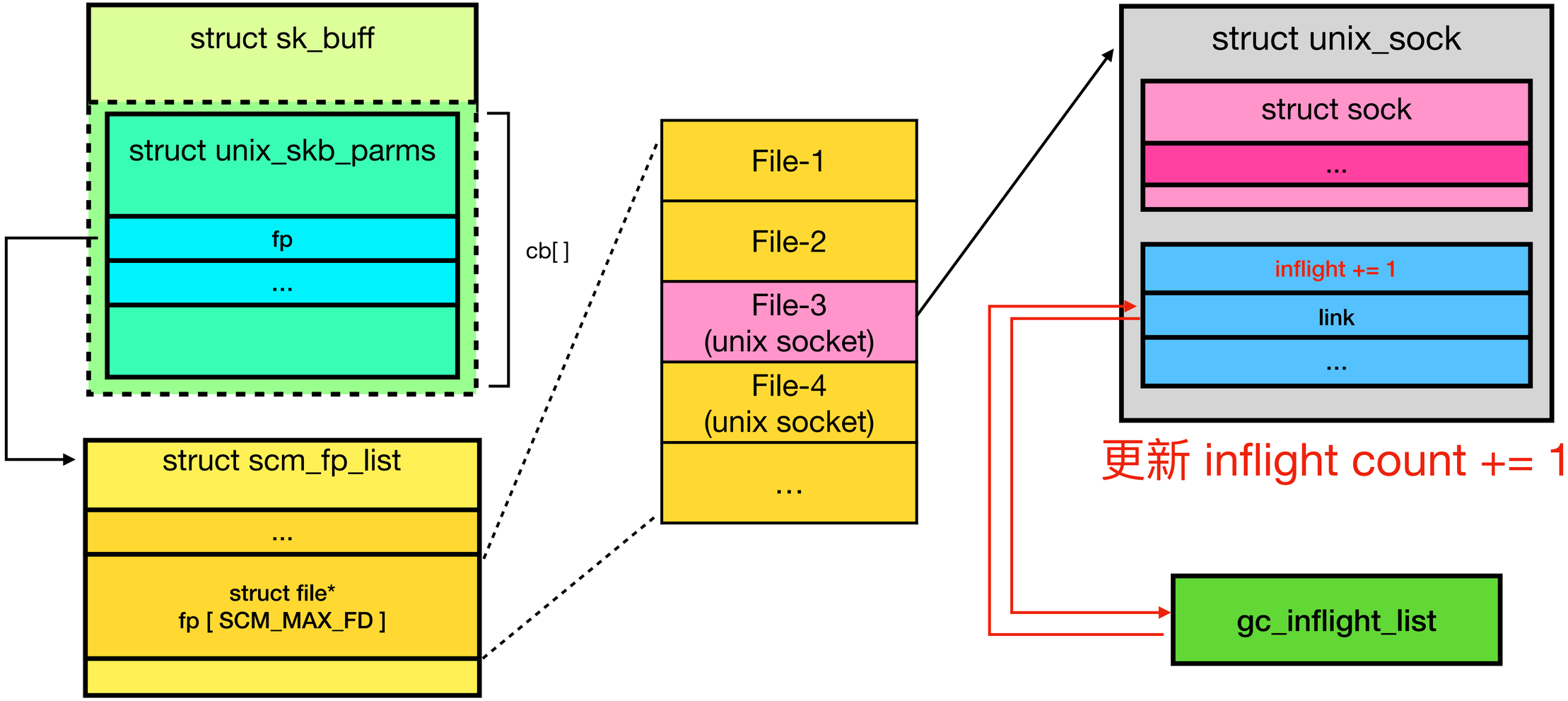
AF_UNIX

unix_stream_ops



AF_UNIX

unix_stream_ops



AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - …
  - [5] Copy data to skb
  - [6] Enqueue to peer’s receive queue
  - [7] Release scm

AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [5] Copy data to skb
    - 透過 `skb_put()` 告訴 skb 實際要用的大小
    - 複製資料到 skb->data or fragment

```
} else {
    skb_put(skb, size - data_len);
    skb->data_len = data_len;
    skb->len = size;
    err = skb_copy_datagram_from_iter(skb, 0, &msg->msg_iter, size);
    if (err) {
        kfree_skb(skb);
        goto out_err;
    }
}
```

AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [6] Enqueue to peer’s receive queue
    - 將 skb enqueue 到 peer socket 的 receive queue
    - Wakeup peer socket

```
skb_queue_tail(&other->sk_receive_queue, skb);
other->sk_data_ready(other);
```

AF_UNIX

unix_stream_ops

- sendmsg (unix_stream_sendmsg)
  - [7] Release scm
    - Attach 時已經 dup 一份 scm object，離開前將舊的釋放掉
      - File object refcnt -= 1

```
void __scm_destroy(struct scm_cookie *scm)
{
    struct scm_fp_list *fpl = scm->fp;
    int i;

    if (fpl) {
        scm->fp = NULL;
        for (i=fpl->count-1; i>=0; i--)
            fput(fpl->fp[i]);
        free_uid(fpl->user);
        kfree(fpl);
    }
}
```

AF_UNIX

unix_stream_ops

- recvmsg (unix_stream_recvmsg)
  - [1] Peek receive queue
  - [2] skb refcnt += 1
  - [3] Call `state->recv_actor()`
    - Call `unix_stream_read_actor()` to copy data
  - [4] skb refcnt -= 1
  - …

```
// [...]
last = skb = skb_peek(&sk->sk_receive_queue);
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
drop_skb = !unix_skb_len(skb);
consume_skb(skb);
// [...]
```

```
static int unix_stream_read_actor(struct sk_buff *skb,
                                  int skip, int chunk,
                                  struct unix_stream_read_state *state)
{
    int ret;

    ret = skb_copy_datagram_msg(skb, UNIXCB(skb).consumed + skip,
                                state->msg, chunk);
    return ret ?: chunk;
}
```

[3]

AF_UNIX

unix_stream_ops

- recvmsg (unix_stream_recvmsg)
  - …
  - [5] Detach files on the skb
  - [6] Unlink skb from receive queue
  - [7] skb refcnt -= 1
  - [8] Install file to fd table

AF_UNIX

unix_stream_ops

- recvmsg (unix_stream_recvmsg)
  - [5] detach files on the skb
    - 遍歷 inflight socket
    - 更新 socket inflight 並嘗試 unlink from gc_inflight_list

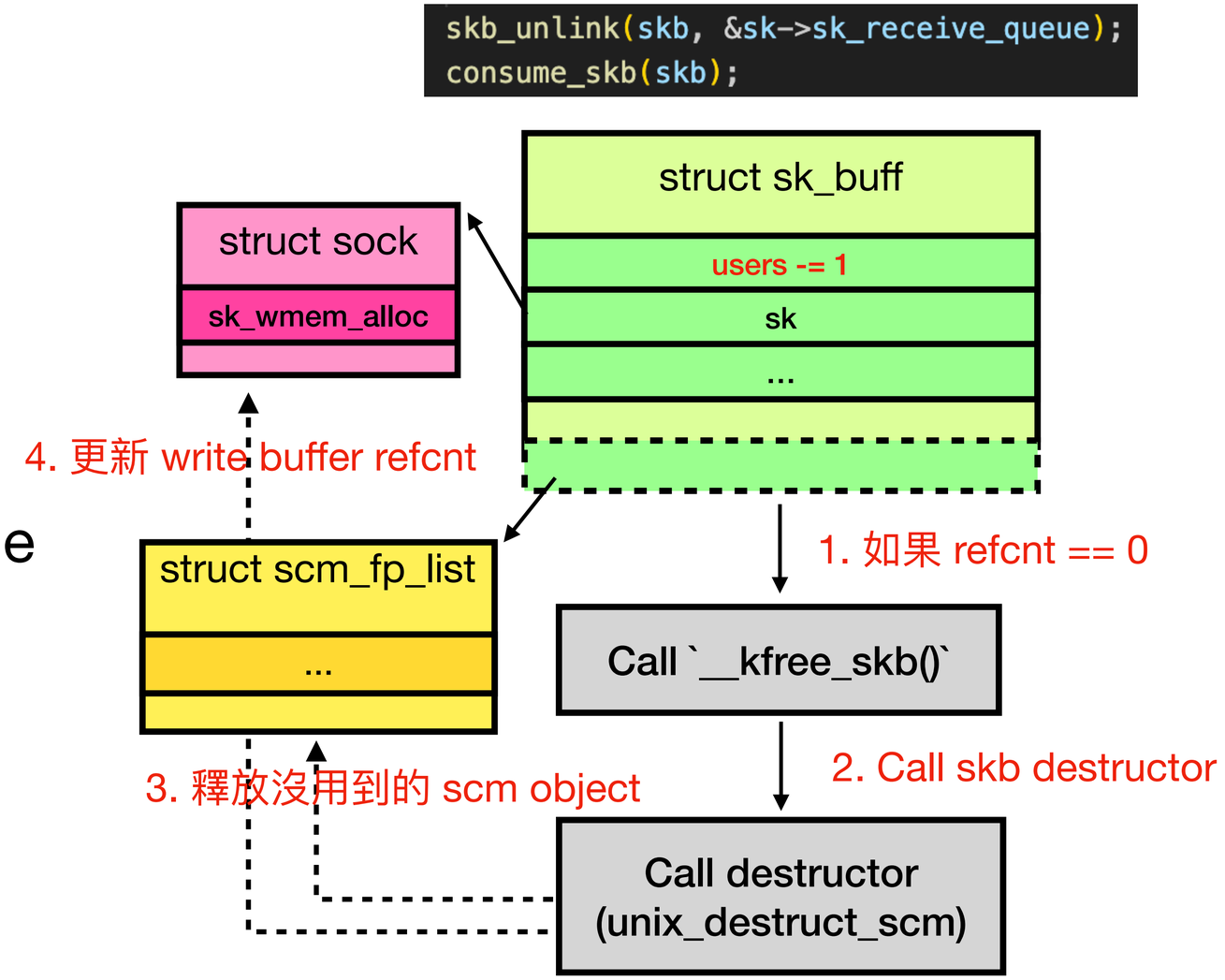
```
for (i = scm->fp->count-1; i >= 0; i--)
    unix_notinflight(scm->fp->user, scm->fp->fp[i]);
```

```
void unix_notinflight(struct user_struct *user, struct file *fp)
{
    struct sock *s = unix_get_socket(fp);

    spin_lock(&unix_gc_lock);

    if (s) {
        struct unix_sock *u = unix_sk(s);

        BUG_ON(!u->inflight);
        BUG_ON(list_empty(&u->link));

        u->inflight--;
        if (!u->inflight)
            list_del_init(&u->link);
```

AF_UNIX

unix_stream_ops

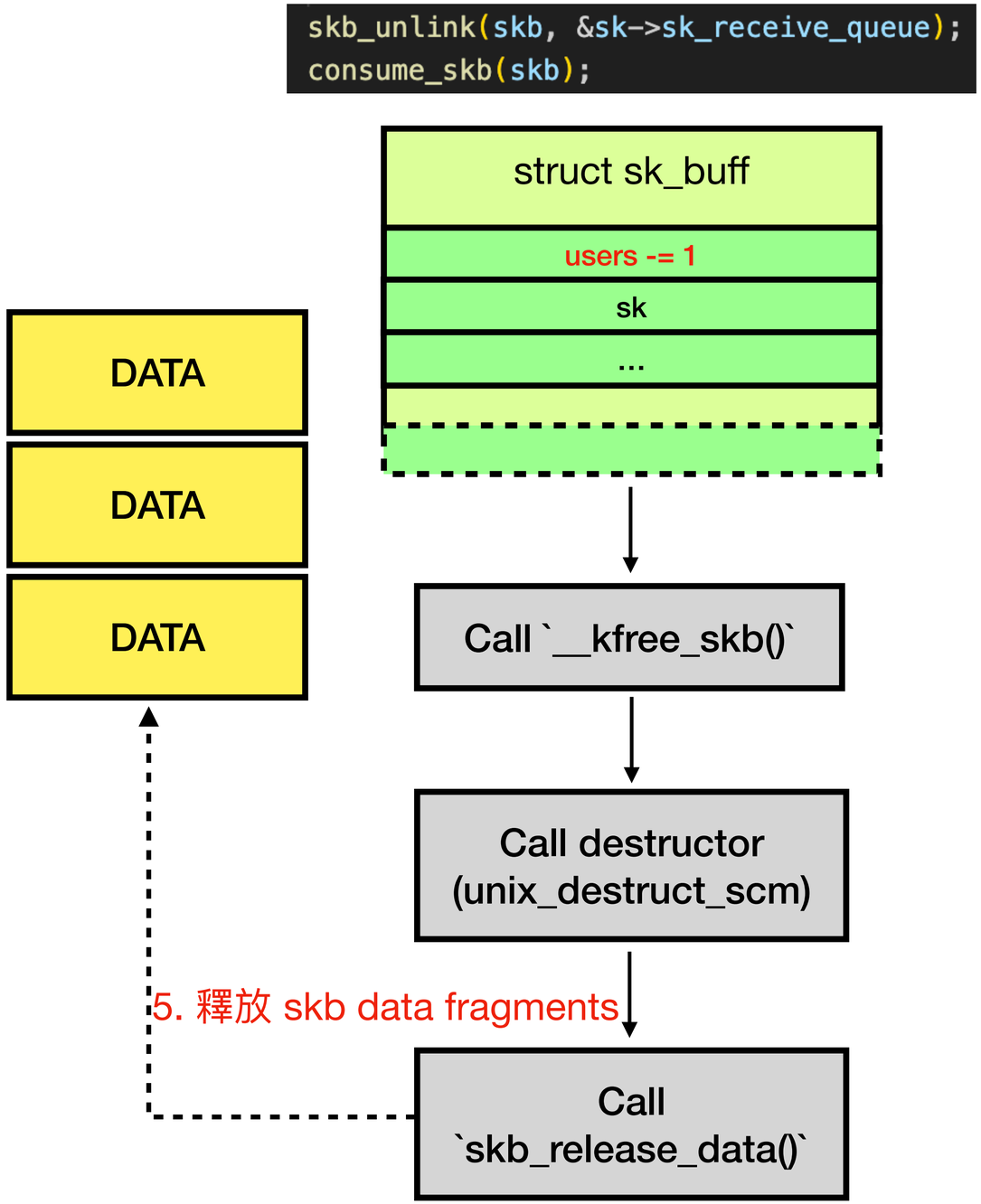
- recvmsg (unix_stream_recvmsg)
  - [6] Unlink skb from receive queue
  - [7] skb refcnt -= 1
    - 更新 skb head



AF_UNIX

unix_stream_ops

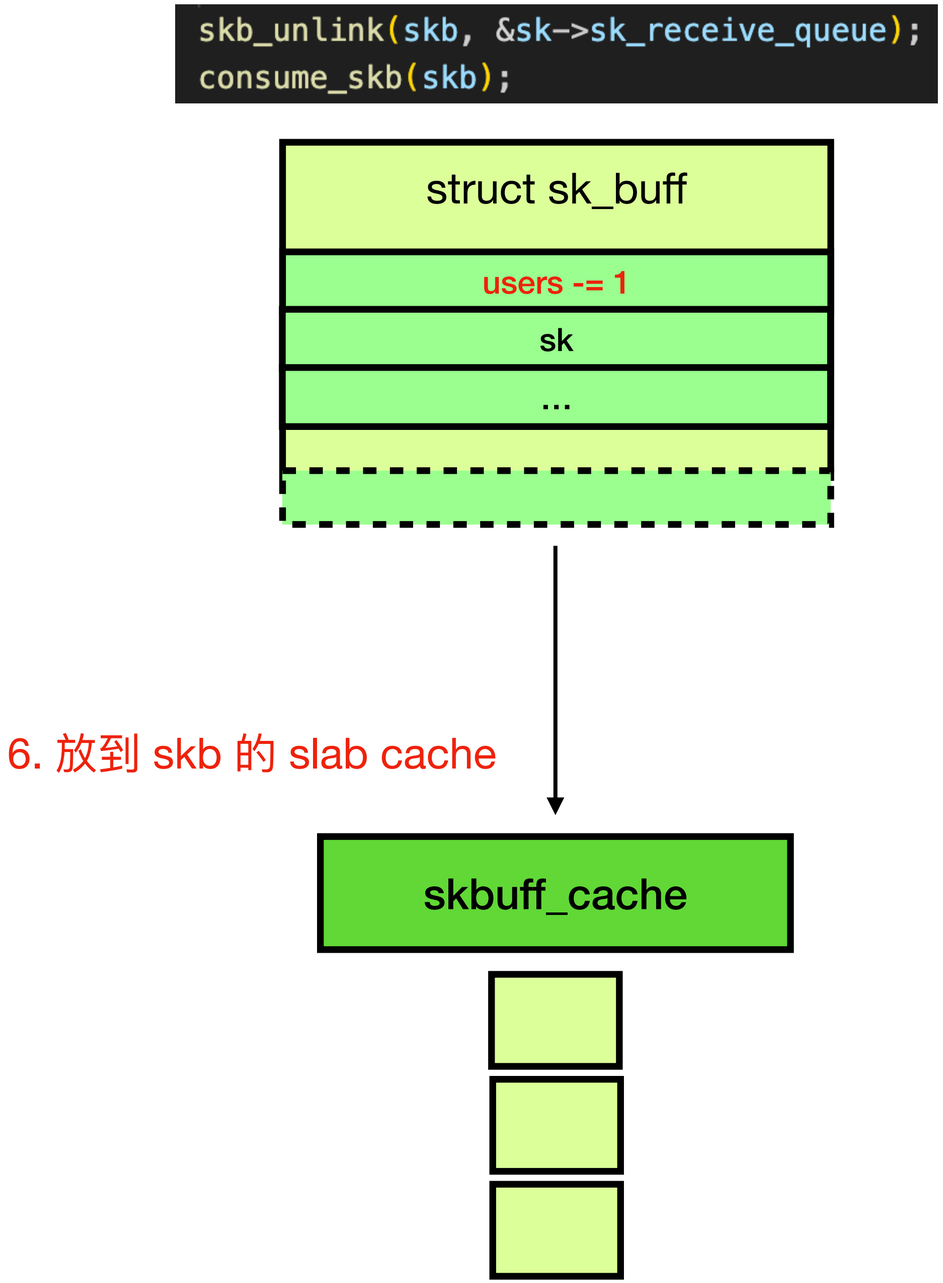
- recvmsg (unix_stream_recvmsg)
  - [6] Unlink skb from receive queue
  - [7] skb refcnt -= 1
    - 更新 skb head
    - 更新 skb data



AF_UNIX

unix_stream_ops

- recvmsg (unix_stream_recvmsg)
  - [6] Unlink skb from receive queue
  - [7] skb refcnt -= 1
    - 更新 skb head
    - 更新 skb data
    - 放到 skbuff_cache



AF_UNIX

unix_stream_ops

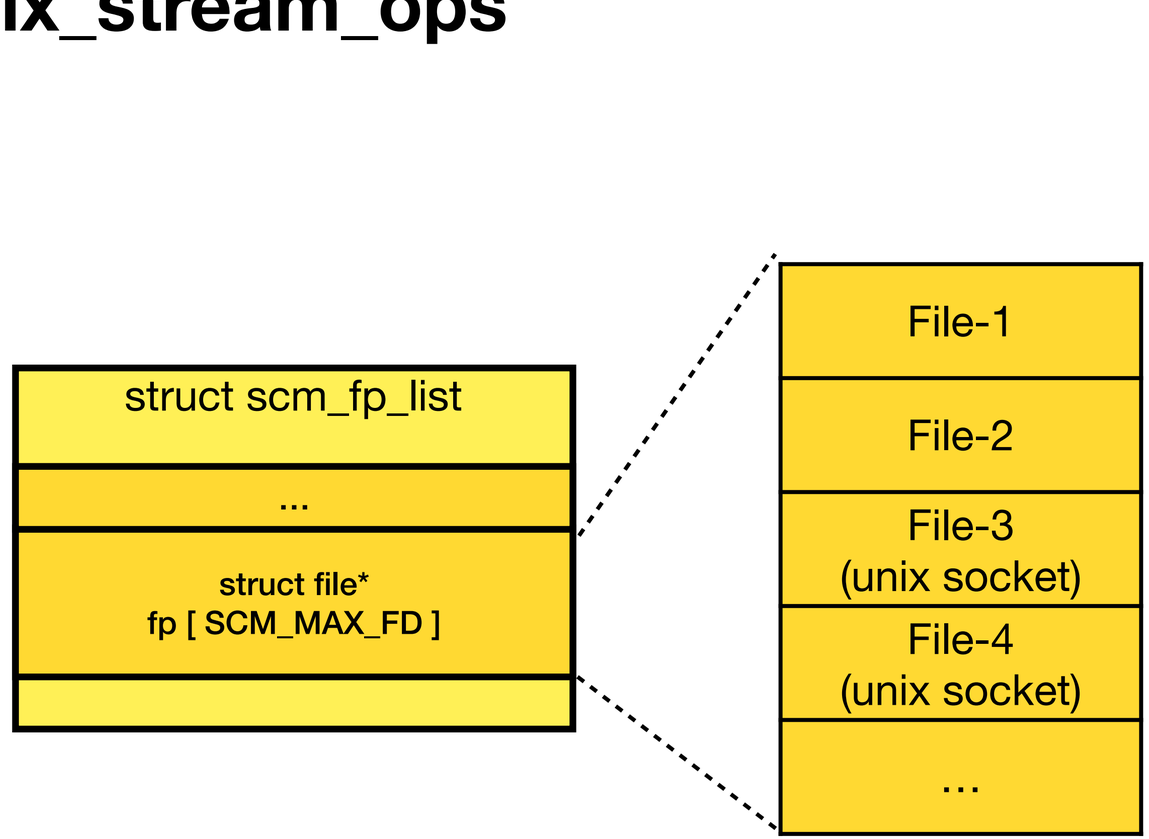
- recvmsg (unix_stream_recvmsg)
  - [8] Install file to fd table
    - 處理 skb 收到的 scm
    - 將 file install 到 process 的 fd table

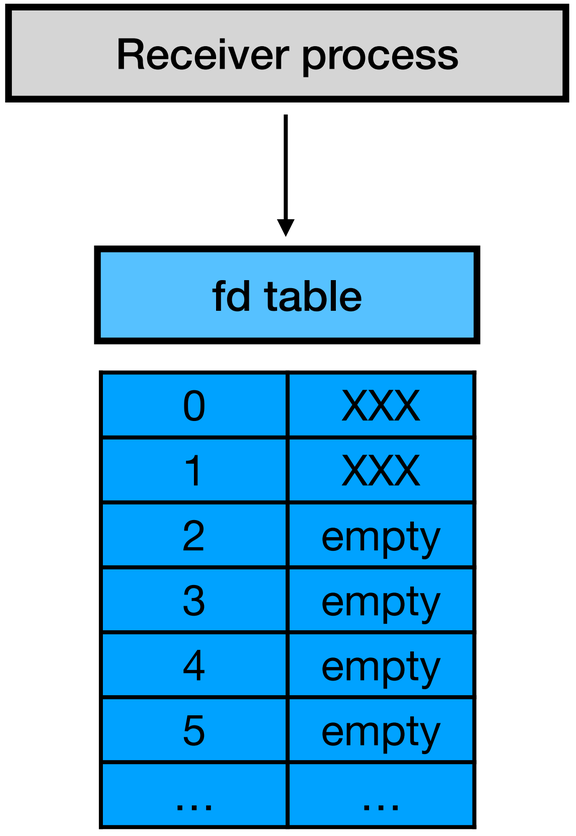
```
for (i = 0; i < fdmax; i++) {
    err = receive_fd_user(scm->fp->fp[i], cmsg_data + i, o_flags);
    if (err < 0)
        break;
}
```

```
int __receive_fd(struct file *file, int __user *ufd, unsigned int o_flags)
{
    int new_fd;
    int error;
    new_fd = get_unused_fd_flags(o_flags);
    // [...]
    fd_install(new_fd, get_file(file));
    __receive_sock(file);
    return new_fd;
}
```

AF_UNIX

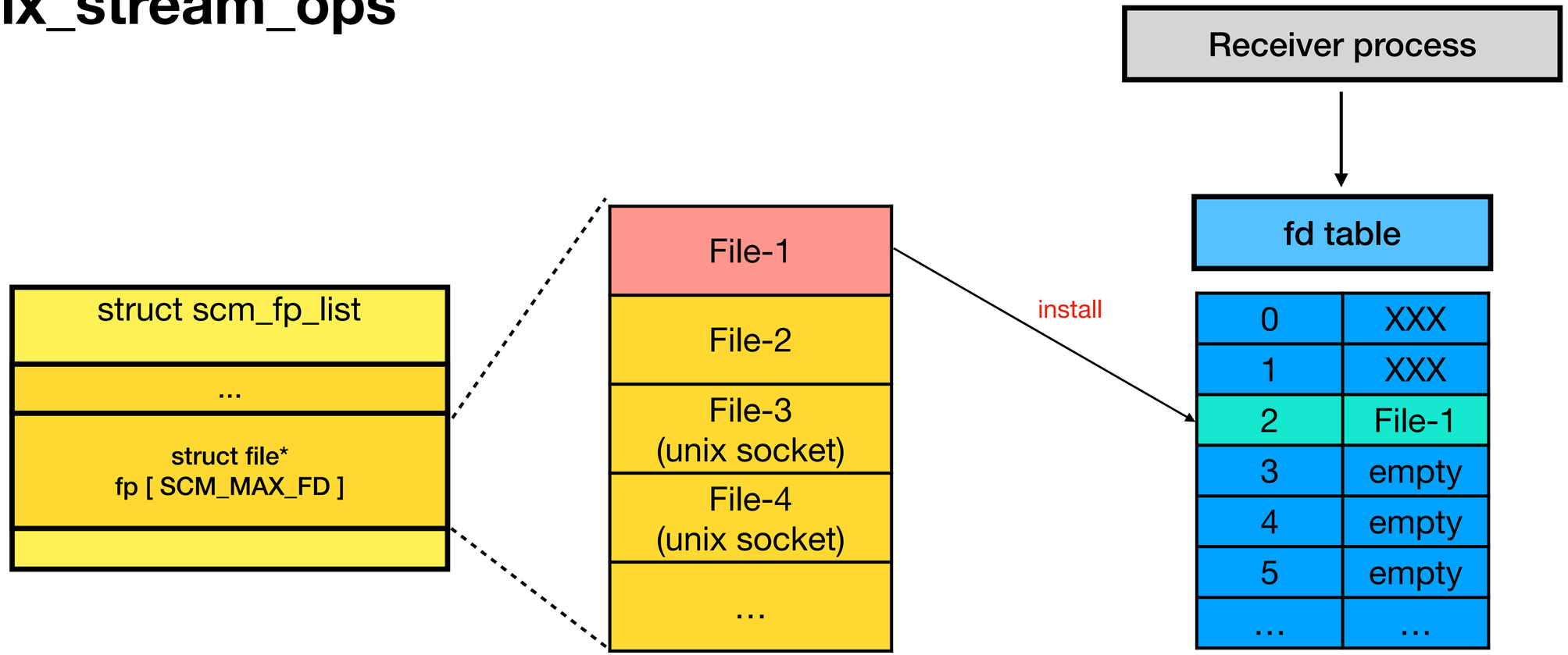
unix_stream_ops





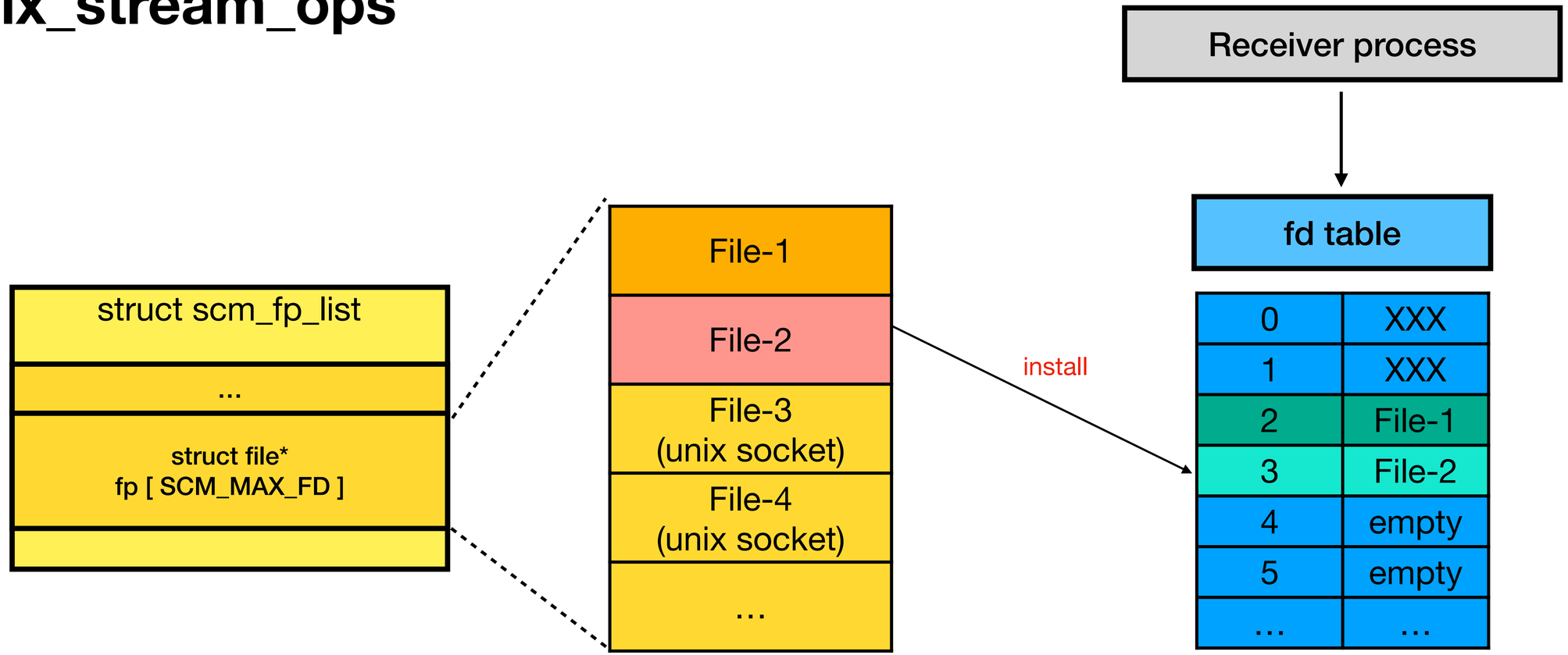
AF_UNIX

unix_stream_ops



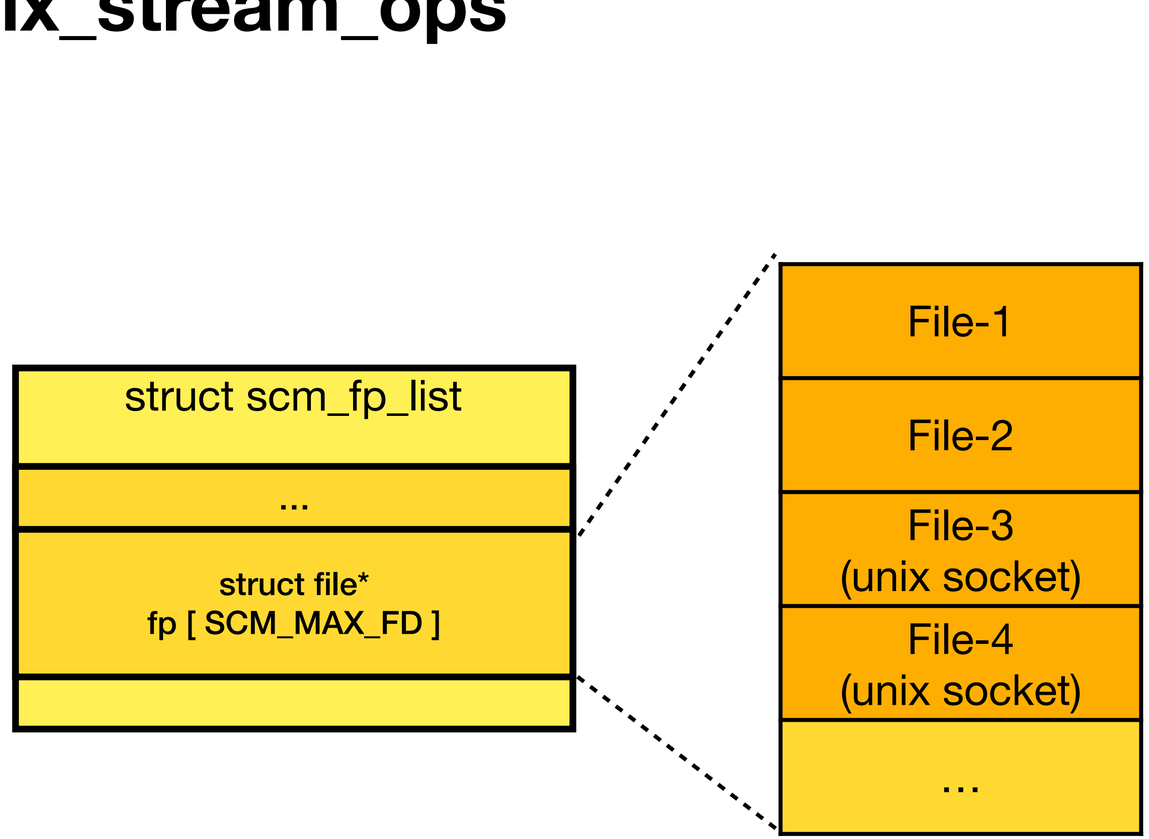
AF_UNIX

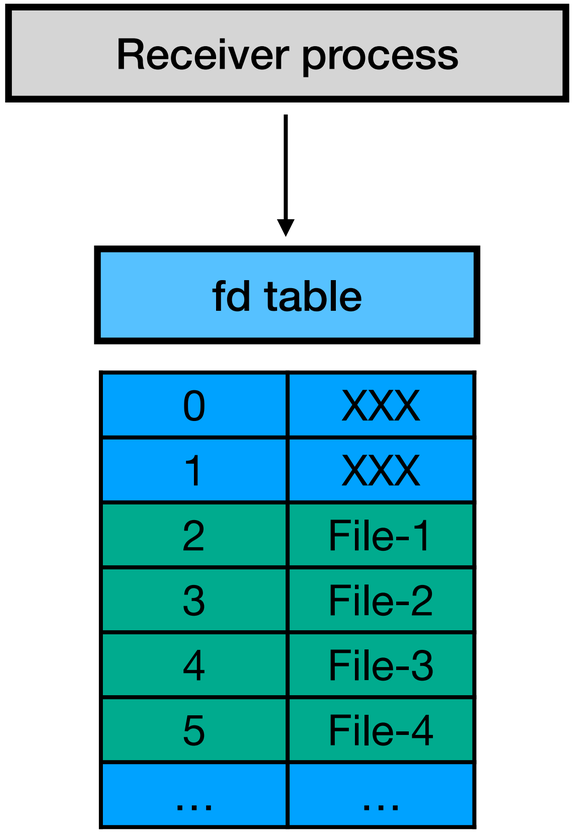
unix_stream_ops



AF_UNIX

unix_stream_ops





AF_UNIX

unix_stream_ops

- close (unix_release)
  - Call protocol close handler
    - Do nothing
  - Call `unix_release_sock()`
    - [1] Update sock state to CLOSE
    - [2] Peer's refcnt -= 1
    - …

```
sk->sk_state = TCP_CLOSE;
```

[1]

```
if (skpair != NULL) {
    if (sk->sk_type == SOCK_STREAM || sk->sk_type == SOCK_SEQPACKET) {
        // [...]
        if (!skb_queue_empty(&sk->sk_receive_queue) || embrion)
            WRITE_ONCE(skpair->sk_err, ECONNRESET);
        // [...]
        skpair->sk_state_change(skpair);
        sk_wake_async(skpair, SOCK_WAKE_WAITD, POLL_HUP);
    }
    unix_dgram_peer_wake_disconnect(sk, skpair);
    sock_put(skpair); /* It may now die */
}
```

[2]

AF_UNIX

unix_stream_ops

- close (unix_release)
  - Call `unix_release_sock()`
    - …
    - [3] 清空 receive queue
      - 最後走到 `__kfree_skb()`
    - [4] 自己 sock 的 refcnt -= 1
    - [5] 呼叫 gc handler `unix_gc()`

```
while ((skb = skb_dequeue(&sk->sk_receive_queue)) != NULL) {
    if (state == TCP_LISTEN)
        unix_release_sock(skb->sk, 1);
    /* passed fds are erased in the kfree_skb hook        */
    UNIXCB(skb).consumed = skb->len;
    kfree_skb(skb);
}
```

```
void __fix_address
kfree_skb_reason(struct sk_buff *skb, enum skb_drop_reason reason)
{
    if (__kfree_skb_reason(skb, reason))
        __kfree_skb(skb);
}
```

[3]

```
sock_put(sk);
// [...]
if (READ_ONCE(unix_tot_inflight))
    unix_gc();      /* Garbage collect fds */
```

[4] [5]

AF_UNIX
Overview

- 共有三種 proto_ops (type)
  - unix_stream_ops
  - unix_dgram_ops
  - unix_seqpacket_ops

```
switch (sock->type) {
case SOCK_STREAM:
    sock->ops = &unix_stream_ops;
    break;
case SOCK_RAW:
    sock->type = SOCK_DGRAM;
    fallthrough;
case SOCK_DGRAM:
    sock->ops = &unix_dgram_ops;
    break;
case SOCK_SEQPACKET:
    sock->ops = &unix_seqpacket_ops;
    break;
default:
    return -ESOCKTNOSUPPORT;
}
```

AF_UNIX

unix_dgram_ops

- ~~socketpair (unix_socketpair)~~
- sendmsg (unix_dgram_sendmsg)
- recvmsg (unix_dgram_recvmsg)
- ~~close (unix_release)~~

AF_UNIX

unix_dgram_ops

- sendmsg (unix_dgram_sendmsg)
  - 大致與 stream 的處理相同，只差在：
    - DGRAM 會拿 peer socket refcnt 但 STREAM 不會

```
} else {
    sunaddr = NULL;
    err = -ENOTCONN;
    other = unix_peer_get(sk);
    if (!other)
        goto out;
}
```

```
out:
    if (other)
        sock_put(other);
    scm_destroy(&scm);
```

DGRAM

```
} else {
    err = -ENOTCONN;
    other = unix_peer(sk);
    if (!other)
        goto out_err;
}
```

```
out_err:
    scm_destroy(&scm);
    return sent ? : err;
```

STREAM

AF_UNIX

unix_dgram_ops

- recvmsg (unix_dgram_recvmsg)
  - [1] 從 receive queue 取得 skb
  - [2] Copy data to user buffer
  - [3] Detach files on the skb
  - [4] Install file to fd table
  - [5] Free datagram skb

跟 stream 的操作大致相同

AF_UNIX

unix_dgram_ops

- recvmsg (unix_dgram_recvmsg)
  - [1] 從 receive queue 取得 skb
  - [2] Copy data to user buffer
  - 只差在 copy data 時沒有更新 skb 的 refcnt
  - [5] Free datagram skb

```
// [...]
last = skb = skb_peek(&sk->sk_receive_queue);
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
drop_skb = !unix_skb_len(skb);
consume_skb(skb);
// [...]
```

```
static int unix_stream_read_actor(struct sk_buff *skb,
                                  int skip, int chunk,
                                  struct unix_stream_read_state *state)
{
    int ret;

    ret = skb_copy_datagram_msg(skb, UNIXCB(skb).consumed + skip,
                                state->msg, chunk);
    return ret ?: chunk;
}
```

STREAM

```
err = skb_copy_datagram_msg(skb, skip, msg, size);
if (err)
    goto out_free;
```

DGRAM

AF_UNIX
Overview

- 共有三種 proto_ops (type)
  - unix_stream_ops
  - unix_dgram_ops
  - unix_seqpacket_ops

```
switch (sock->type) {
case SOCK_STREAM:
    sock->ops = &unix_stream_ops;
    break;
case SOCK_RAW:
    sock->type = SOCK_DGRAM;
    fallthrough;
case SOCK_DGRAM:
    sock->ops = &unix_dgram_ops;
    break;
case SOCK_SEQPACKET:
    sock->ops = &unix_seqpacket_ops;
    break;
default:
    return -ESOCKTNOSUPPORT;
}
```

AF_UNIX

unix_seqpacket_ops

- sendmsg (unix_seqpacket_sendmsg)
- recvmsg (unix_seqpacket_recvmsg)
- 最後都是走 DGRAM 的 handler

```
static int unix_seqpacket_sendmsg(struct socket *sock, struct msghdr *msg,
                                  size_t len)
{
    int err;
    struct sock *sk = sock->sk;
    // [...]
    return unix_dgram_sendmsg(sock, msg, len);
}
```

sendmsg

```
static int unix_seqpacket_recvmsg(struct socket *sock, struct msghdr *msg,
                                  size_t size, int flags)
{
    struct sock *sk = sock->sk;
    // [...]
    return unix_dgram_recvmsg(sock, msg, size, flags);
}
```

recvmsg

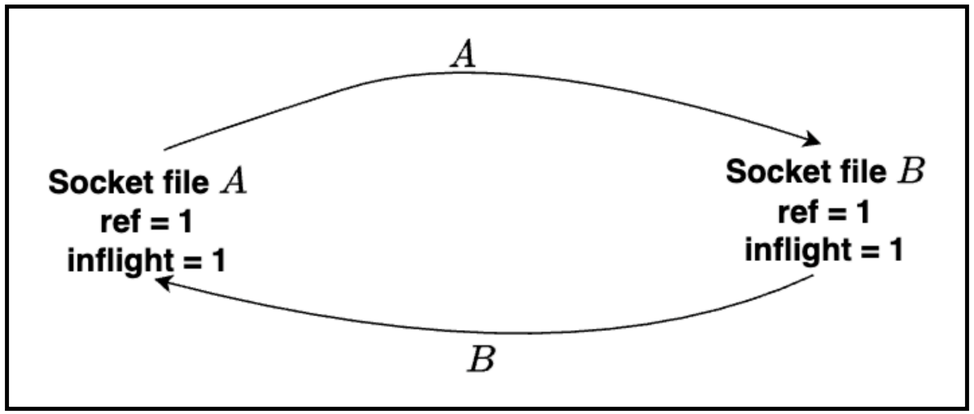
AF_UNIX

Garbage Collection

- Garbage collector - 用來回收 inflight sock object

AF_UNIX
Garbage Collection

- Garbage collector - 用來回收 inflight sock object
  - A -{A}-> B ; B -{B}-> A
  - Close A
  - Close B



https://googleprojectzero.blogspot.com/2022/08/the-quantum-state-of-linux-kernel.html

AF_UNIX

Garbage Collection

- Garbage collector - 用來回收 inflight sock object
  - [1] 檢查並設 `gc_in_progress = true` 代表正在執行
    - 如果已經為 true 就直接離開
  - [2] 遍歷所有在 gc_inflight_list 上的 sock
    - 如果 file 的 refcnt 與 inflight 相同就移到 gc_candidates
    - 沒有 process 能存取到時會滿足 (?)

```
spin_lock(&unix_gc_lock);

/* Avoid a recursive GC. */
if (gc_in_progress)
    goto out;

WRITE_ONCE(gc_in_progress, true);
```

[1]

```
list_for_each_entry_safe(u, next, &gc_inflight_list, link) {
    struct sock *sk = &u->sk;
    long total_refs;

    total_refs = file_count(sk->sk_socket->file);
    if (total_refs == u->inflight) {
        list_move_tail(&u->link, &gc_candidates);
        __set_bit(UNIX_GC_CANDIDATE, &u->gc_flags);
        __set_bit(UNIX_GC_MAYBE_CYCLE, &u->gc_flags);

        if (sk->sk_state == TCP_LISTEN) {
            unix_state_lock_nested(sk, U_LOCK_GC_LISTENER);
            unix_state_unlock(sk);
        }
    }
}
```

[2]

AF_UNIX
Garbage Collection

- Garbage collector - 用來回收 inflight sock object
  - [3] 遍歷所有在 gc_candidates 上的 sock
    - 遍歷 sock 的 receive queue
      - 如果這些 skb 也有其他 sock file 也在 gc_candidates 當中，更新其 inflight

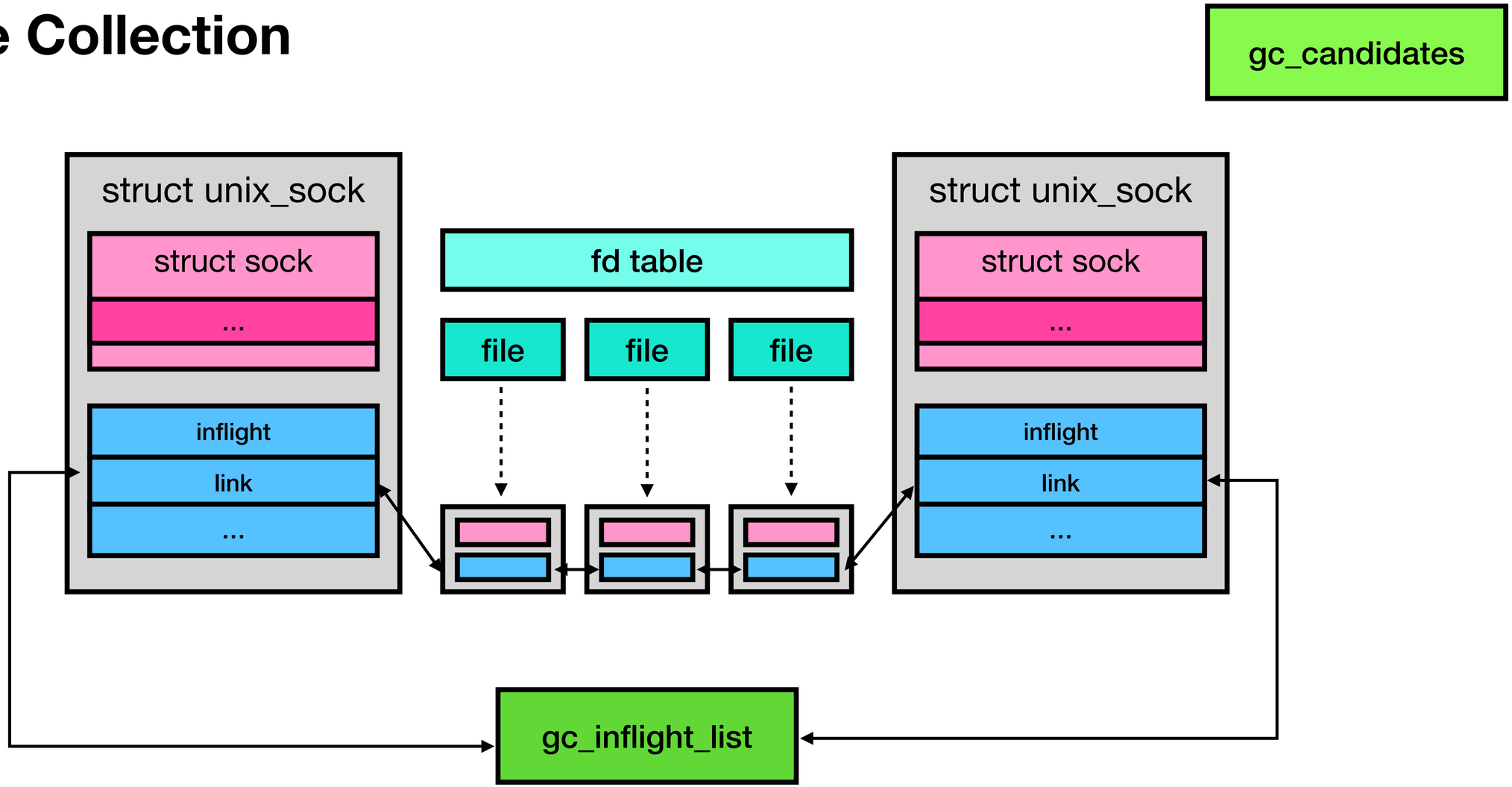
```
int nfd = UNIXCB(skb).fp->count;
struct file **fp = UNIXCB(skb).fp->fp;
while (nfd--) {
    struct sock *sk = unix_get_socket(*fp++);
    if (sk) {
        struct unix_sock *u = unix_sk(sk);
        if (test_bit(UNIX_GC_CANDIDATE, &u->gc_flags)) {
            hit = true;

            func(u);
        }
    }
}
```

```
static void dec_inflight(struct unix_sock *usk)
{
    usk->inflight--;
}
```

  - P.S. inflight 可以當作被 skb reference 到的次數

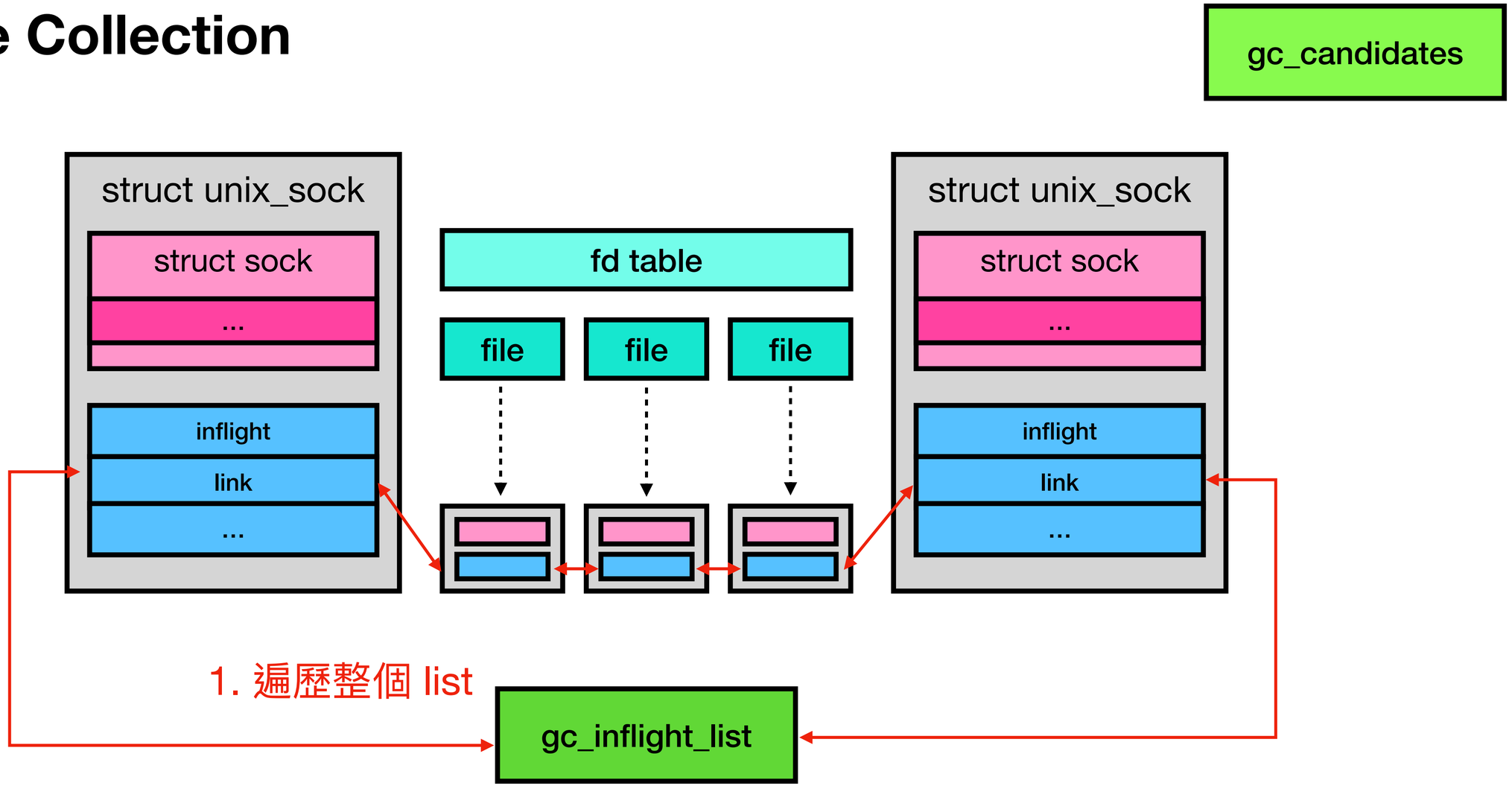
AF_UNIX

Garbage Collection



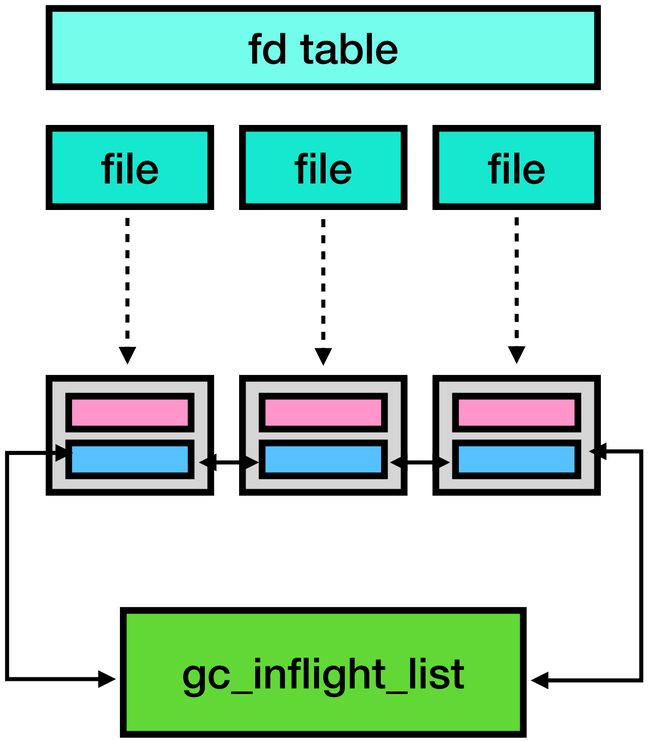
AF_UNIX

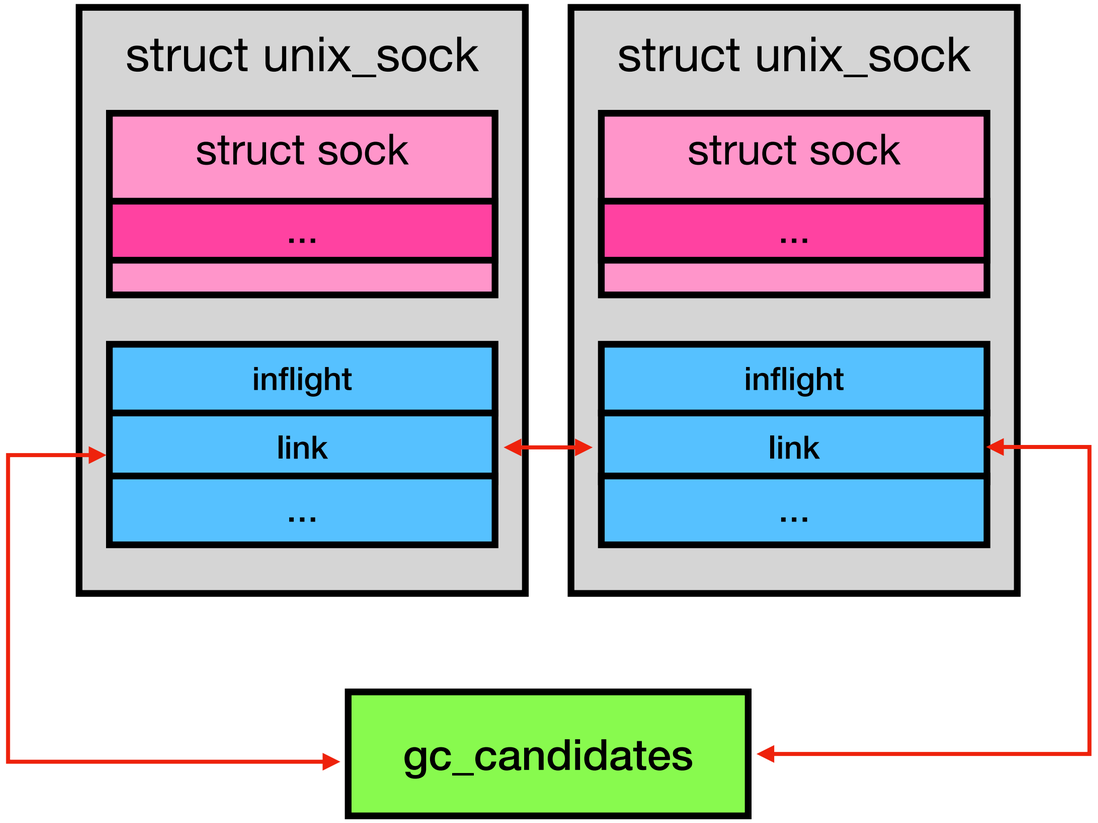
Garbage Collection



AF_UNIX

Garbage Collection

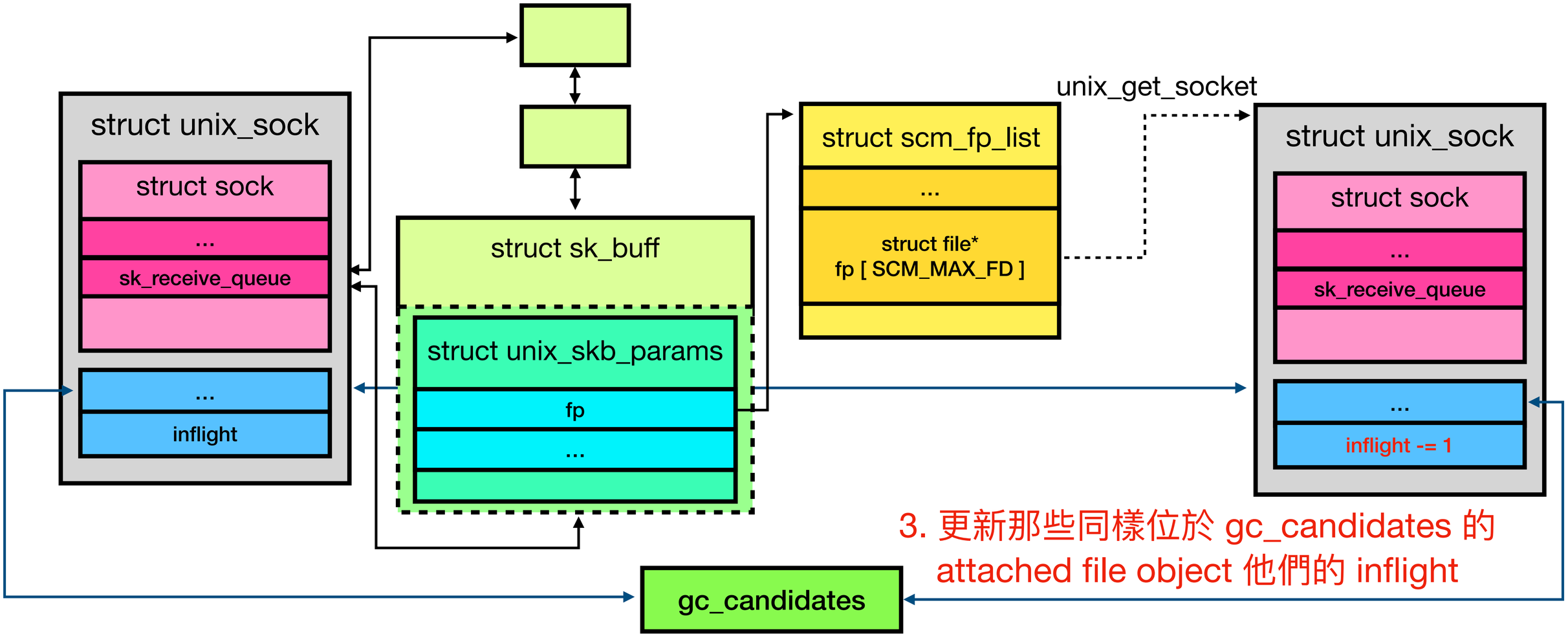




2. 沒外部 file reference 會被移動到 candidate list

AF_UNIX

Garbage Collection



AF_UNIX
Garbage Collection

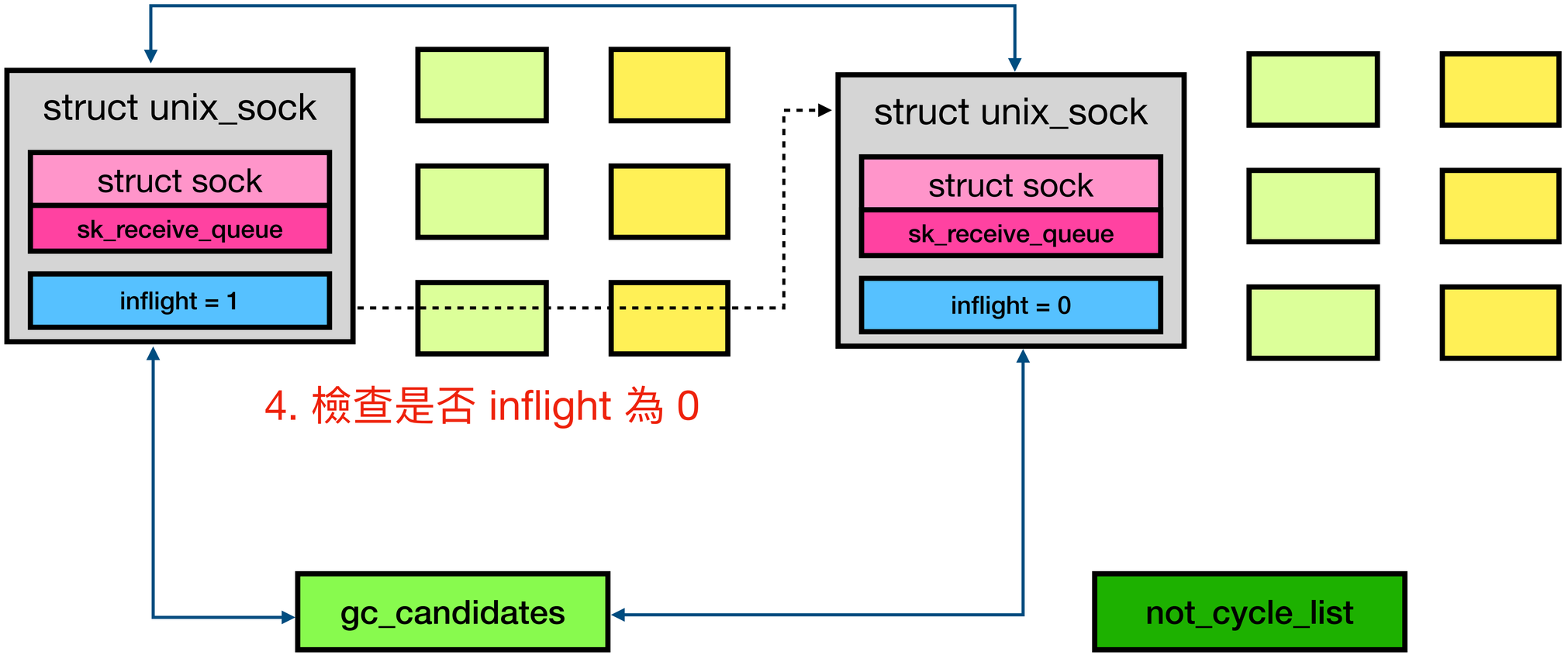
- Garbage collector - 用來回收 inflight sock object
  - [4] 再次遍歷所有在 gc_candidates 上的 sock
    - 如果 inflight != 0，就移到 not_cycle_list
      - 不為 0 代表 gc 完還是會被其他 skb 存取到
    - 遍歷 sock 的 receive queue 並恢復 skb inflight
      - 同時仍嘗試把 sock 加到 gc_candidates (?)

```
while (cursor.next != &gc_candidates) {
    u = list_entry(cursor.next, struct unix_sock, link);

    /* Move cursor to after the current position. */
    list_move(&cursor, &u->link);

    if (u->inflight) {
        list_move_tail(&u->link, &not_cycle_list);
        __clear_bit(UNIX_GC_MAYBE_CYCLE, &u->gc_flags);
        scan_children(&u->sk, inc_inflight_move_tail, NULL);
    }
}
```

```
static void inc_inflight_move_tail(struct unix_sock *u)
{
    u->inflight++;
    if (test_bit(UNIX_GC_MAYBE_CYCLE, &u->gc_flags))
        list_move_tail(&u->link, &gc_candidates);
}
```

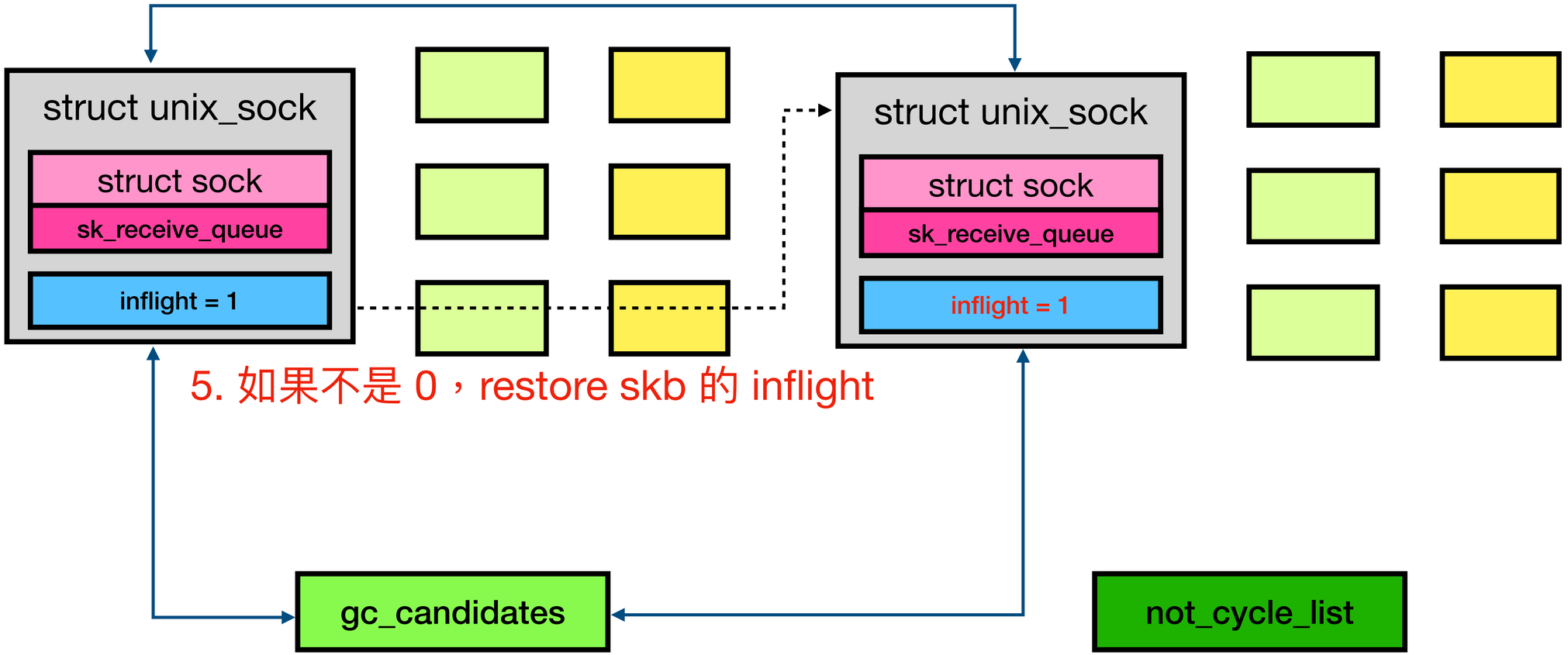
AF_UNIX

Garbage Collection



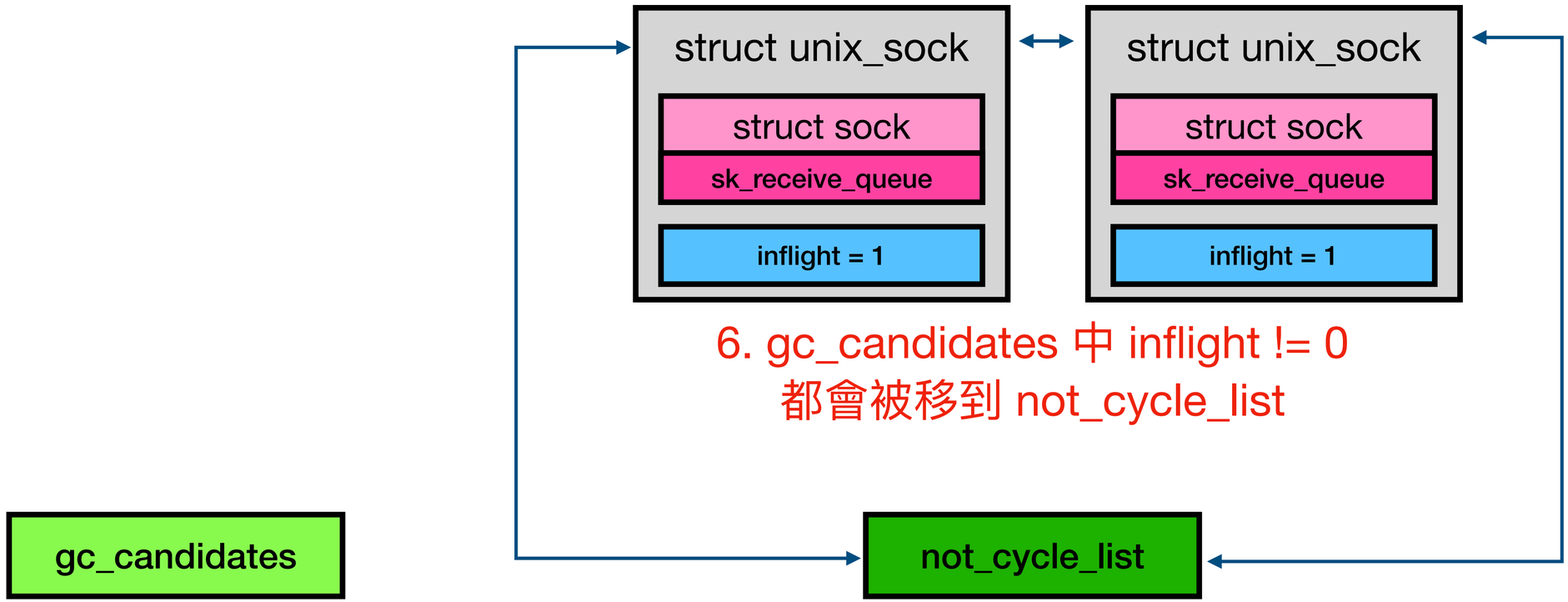
AF_UNIX

Garbage Collection



AF_UNIX

Garbage Collection



AF_UNIX

Garbage Collection

- Garbage collector - 用來回收 inflight sock object
  - [5] 取出 gc_candidates 上 sock 的所有 skb in receive queue 到 hitlist
  - [6] 移動 not_cycle_list 到 gc_inflight_list
  - [7] purge hit list
  - [8] 移動 gc_candidates 到 gc_inflight_list

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
```

[5]

```
while (!list_empty(&not_cycle_list)) {
    u = list_entry(not_cycle_list.next, struct unix_sock, link);
    __clear_bit(UNIX_GC_CANDIDATE, &u->gc_flags);
    list_move_tail(&u->link, &gc_inflight_list);
}
```

[6]

```
__skb_queue_purge(&hitlist);
while ((skb = __skb_dequeue(list)) != NULL)
    kfree_skb_reason(skb, reason);
```

[7]

```
list_for_each_entry_safe(u, next, &gc_candidates, link)
    list_move_tail(&u->link, &gc_inflight_list);
```

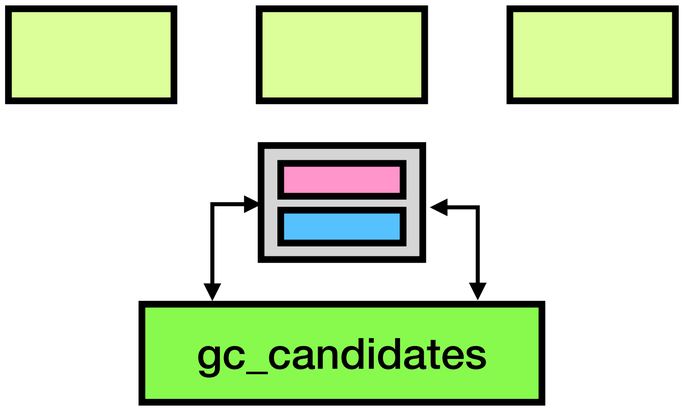
[8]

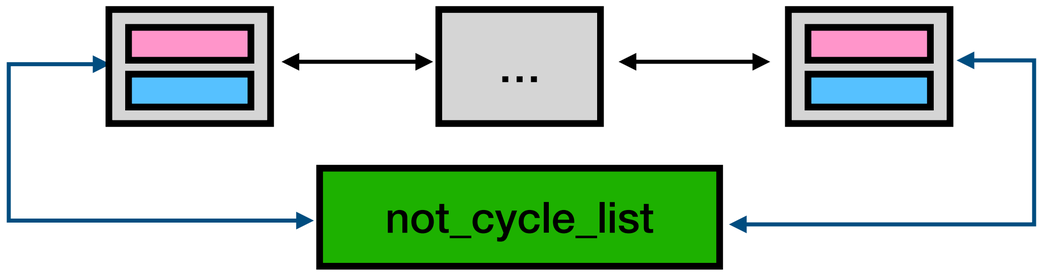
AF_UNIX

Garbage Collection

gc_inflight_list

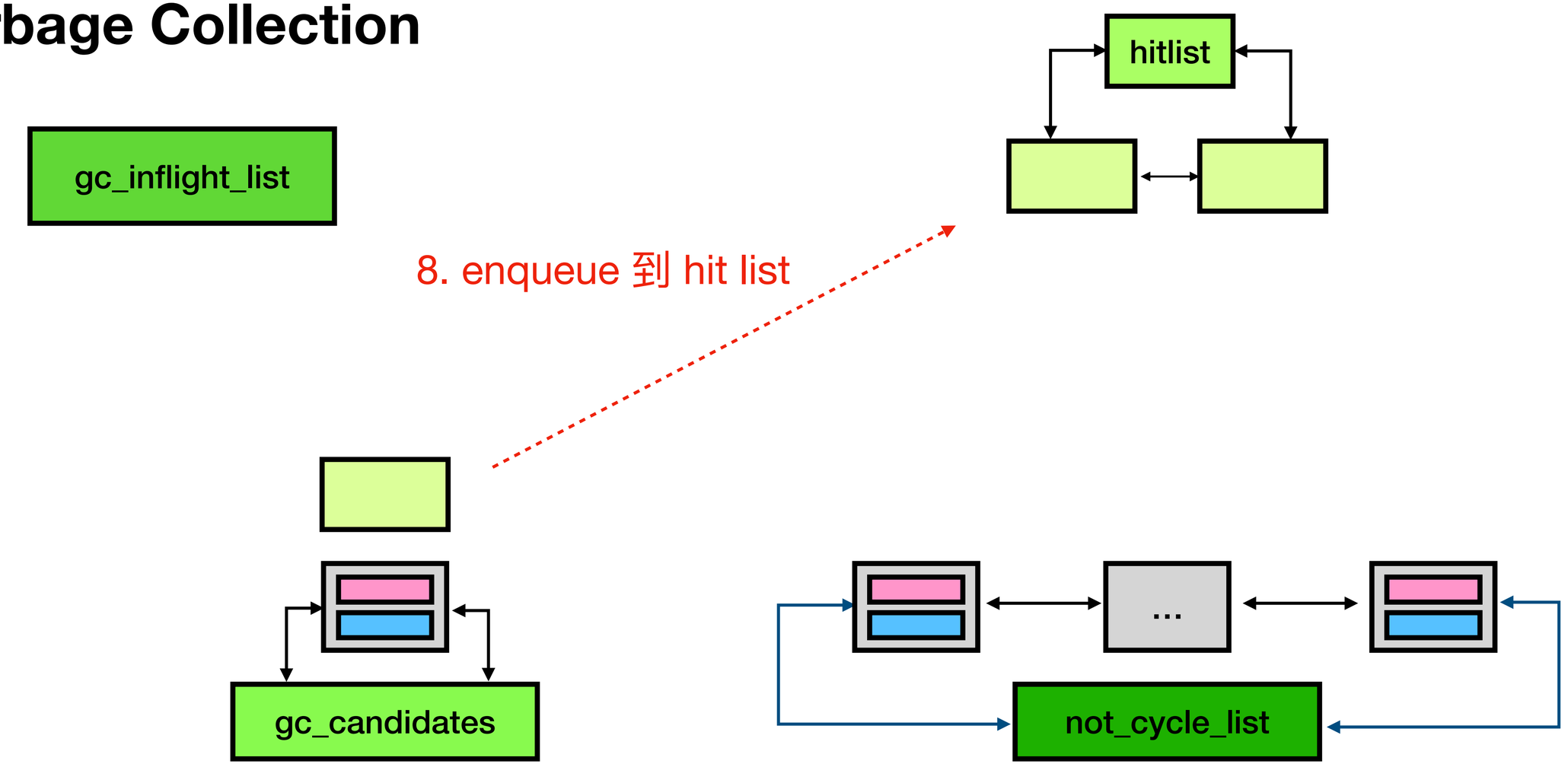
7. 這些 sock 包含了沒被使用 (inflight) 其他的 sock 的 skb





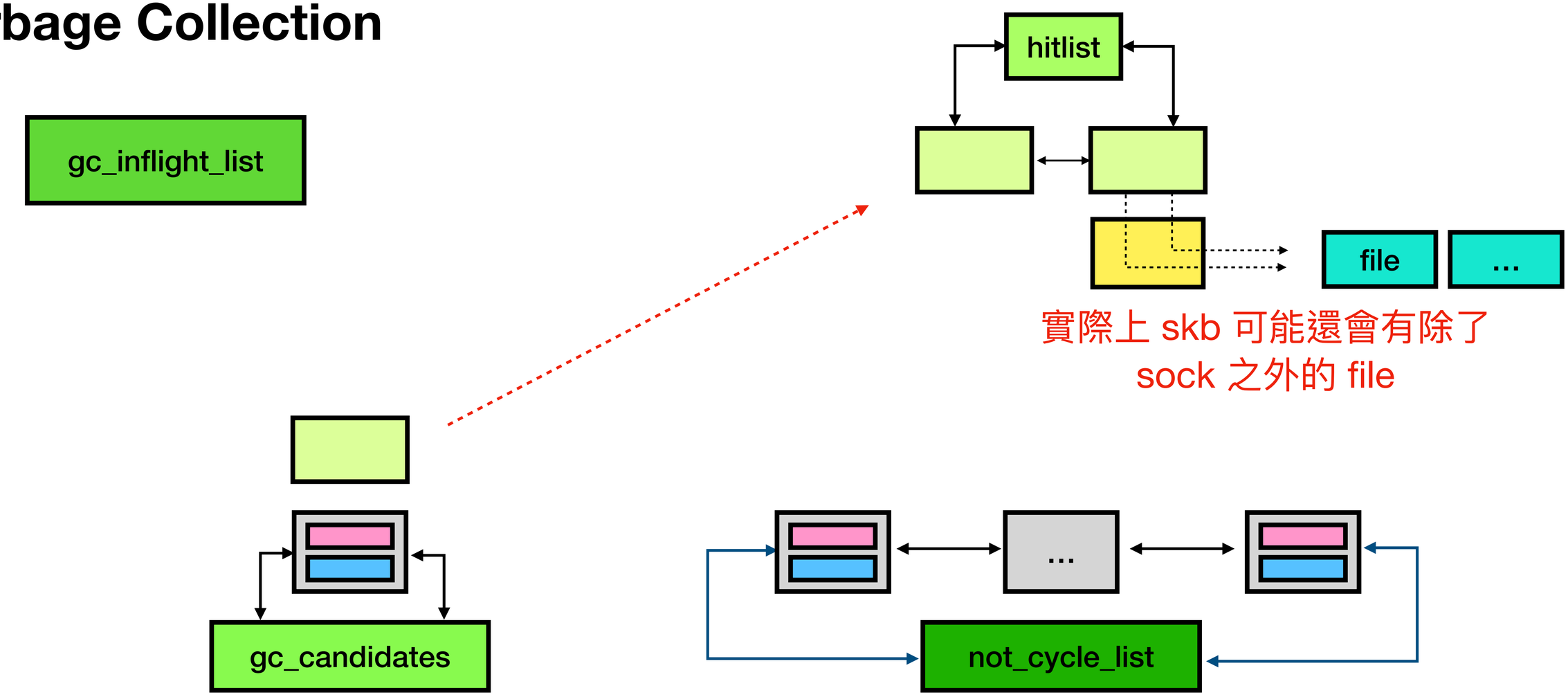
AF_UNIX

Garbage Collection



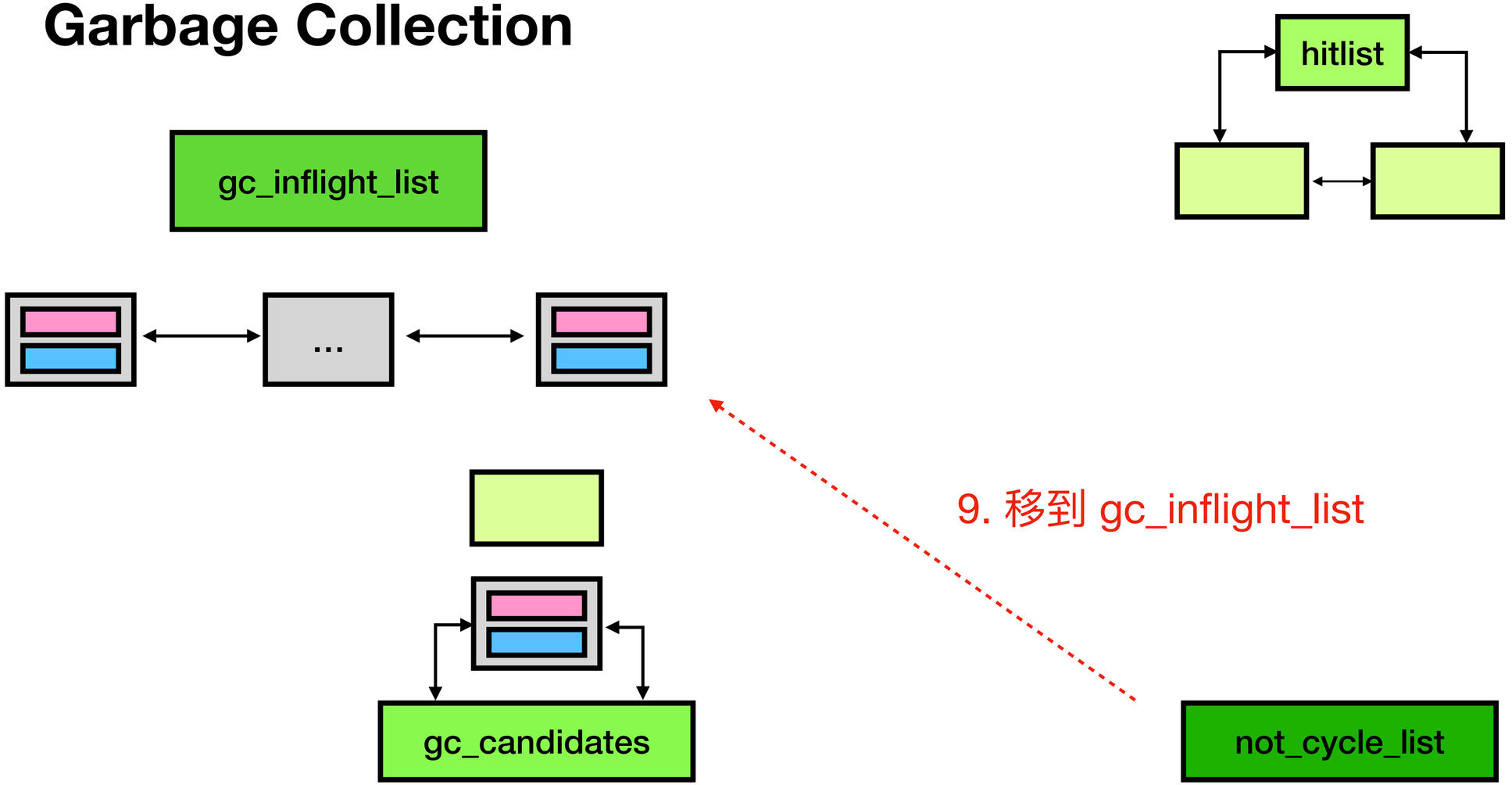
AF_UNIX

Garbage Collection



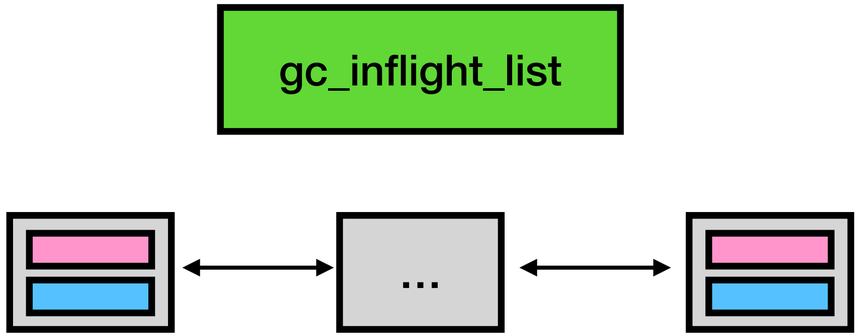
AF_UNIX

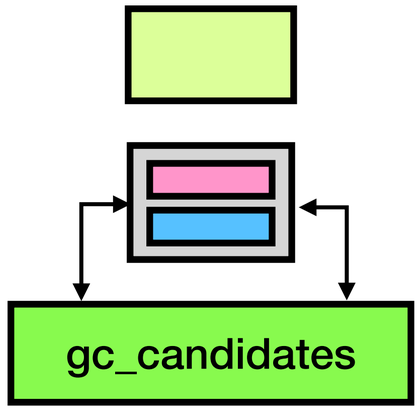
Garbage Collection

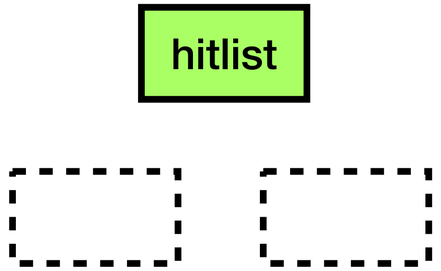


AF_UNIX

Garbage Collection



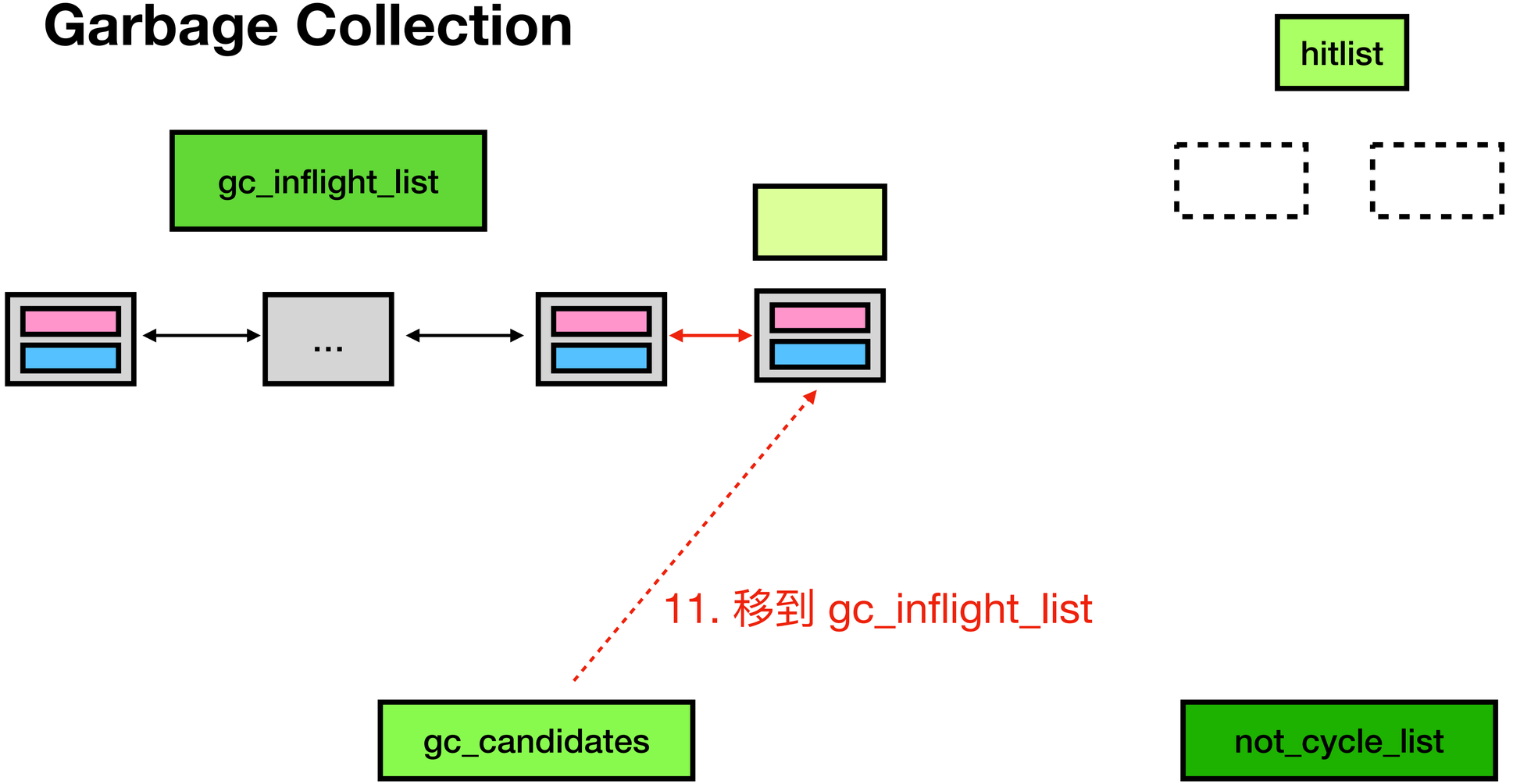




10. purge hit list

AF_UNIX

Garbage Collection



AF_UNIX
Vulnerability

- net: unix: properly re-increment inflight counter of GC discarded candidates (170314)
- af_unix: fix garbage collect vs MSG_PEEK (210728)
- fget: check that the fd still exists after getting a ref to it (211203)
- af_unix: Fix null-ptr-deref in unix_stream_sendpage() (230821)
- af_unix: Fix garbage collector racing against connect() (240409)
- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (240516)

AF_UNIX

170314

- net: unix: properly re-increment inflight counter of GC discarded candidates
  - GC 流程
    - [1] decrements their "children's" inflight counter
    - [2] checks which inflight counters are now 0
    - [3] increments all inflight counters back

```
list_for_each_entry(u, &gc_candidates, link)
    scan_children(&u->sk, dec_inflight, NULL);                    [1]

list_add(&cursor, &gc_candidates);
while (cursor.next != &gc_candidates) {
    u = list_entry(cursor.next, struct unix_sock, link);
    list_move(&cursor, &u->link);
    if (u->inflight) {
        list_move_tail(&u->link, &not_cycle_list);
        __clear_bit(UNIX_GC_MAYBE_CYCLE, &u->gc_flags);
        scan_children(&u->sk, inc_inflight_move_tail, NULL);
    }
}
list_del(&cursor);                                                [2]

while (!list_empty(&not_cycle_list)) {
    u = list_entry(not_cycle_list.next, struct unix_sock, link);
    __clear_bit(UNIX_GC_CANDIDATE, &u->gc_flags);
    list_move_tail(&u->link, &gc_inflight_list);
}

skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {                    [3]
    scan_children(&u->sk, inc_inflight, &hitlist);
}
```

AF_UNIX
170314

- net: unix: properly re-increment inflight counter of GC discarded candidates
  - [1] Callback function 會檢查 UNIX_GC_CANDIDATE flag
  - [2] 因此要在所有 inflight 都恢復到原本的狀態時在 unset flag

```
if (test_bit(UNIX_GC_CANDIDATE, &u->gc_flags)) {
    hit = true;
    func(u);
}
```

[1]

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
}

while (!list_empty(&not_cycle_list)) {
    u = list_entry(not_cycle_list.next, struct unix_sock, link);
    __clear_bit(UNIX_GC_CANDIDATE, &u->gc_flags);
    list_move_tail(&u->link, &gc_inflight_list);
}
```

[2]

AF_UNIX
170314

- net: unix: properly re-increment inflight counter of GC discarded candidates
  - Patch
    - 移動取得 skb 的 code 位置，從 [2] 改成 [1]

[1]
[2]

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
}

while (!list_empty(&not_cycle_list)) {
    u = list_entry(not_cycle_list.next, struct unix_sock, link);
    __clear_bit(UNIX_GC_CANDIDATE, &u->gc_flags);
    list_move_tail(&u->link, &gc_inflight_list);
}

// skb_queue_head_init(&hitlist);
// list_for_each_entry(u, &gc_candidates, link) {
//  scan_children(&u->sk, inc_inflight, &hitlist);
// }
```

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - GC 正在釋放 socket-A 的 skb 的過程中

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
__skb_queue_purge(&hitlist);
```

```
if (hit && hitlist != NULL) {
    __skb_unlink(skb, &x->sk_receive_queue);
    __skb_queue_tail(hitlist, skb);
}
```

  - 同時也在 recvmsg(socket-A)

```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

```
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
```

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - GC 正在釋放 socket-A 的 skb 的過程中

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
__skb_queue_purge(&hitlist);
```

```
if (hit && hitlist != NULL) {
    __skb_unlink(skb, &x->sk_receive_queue);
    __skb_queue_tail(hitlist, skb);
}
```

  - 同時也在 recvmsg(socket-A)

Peek 不會增加 refcnt

```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

```
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
```

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - [1] Unix GC 會 hold unix_gc_lock，並預期 inflight socket file 的 f_count 與 inflight 都不會改變
  - [2] recvmsg detach skb 的 inflight socket file 時會等 unix_gc_lock

```
spin_lock(&unix_gc_lock);
// [...]
list_for_each_entry_safe(u, next, &gc_inflight_list, link) {
    struct sock *sk = &u->sk;
    long total_refs;

    total_refs = file_count(sk->sk_socket->file);

    BUG_ON(!u->inflight);
    BUG_ON(total_refs < u->inflight);
    if (total_refs == u->inflight) {
        list_move_tail(&u->link, &gc_candidates);
```

[1]

```
void unix_notinflight(struct user_struct *user, struct file *fp)
{
    struct sock *s = unix_get_socket(fp);

    spin_lock(&unix_gc_lock);
    if (s) {
        struct unix_sock *u = unix_sk(s);
        // [...]
        u->inflight--;
        if (!u->inflight)
            list_del_init(&u->link);
```

[2]

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - [3] detach 完才會 install file，process 才能存取到這些 inflight socket file

```
int __receive_fd(struct file *file, int __user *ufd, unsigned int o_flags)
{
    int new_fd;
    int error;
    // [...]
    fd_install(new_fd, get_file(file));
    __receive_sock(file);
    return new_fd;
}
```

[3]

AF_UNIX
210728

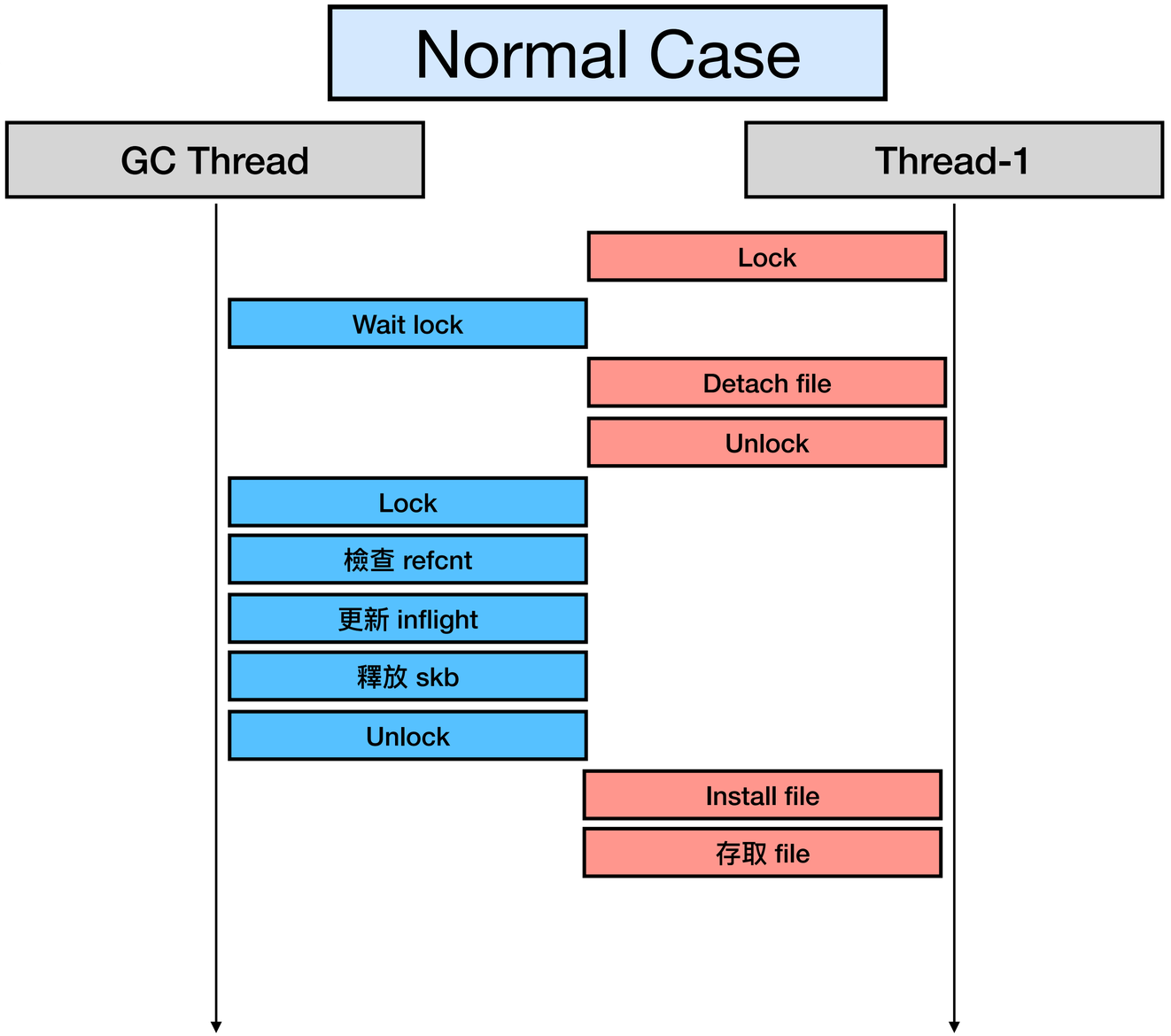
- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - [3] detach 完才會 install file，process 才能存取到這些 inflight socket file
  - 如果沒有上 lock 會發生什麼事？ 🤔

```c
int __receive_fd(struct file *file, int __user *ufd, unsigned int o_flags)
{
    int new_fd;
    int error;
    // [...]
    fd_install(new_fd, get_file(file));
    __receive_sock(file);
    return new_fd;
}
```

[3]

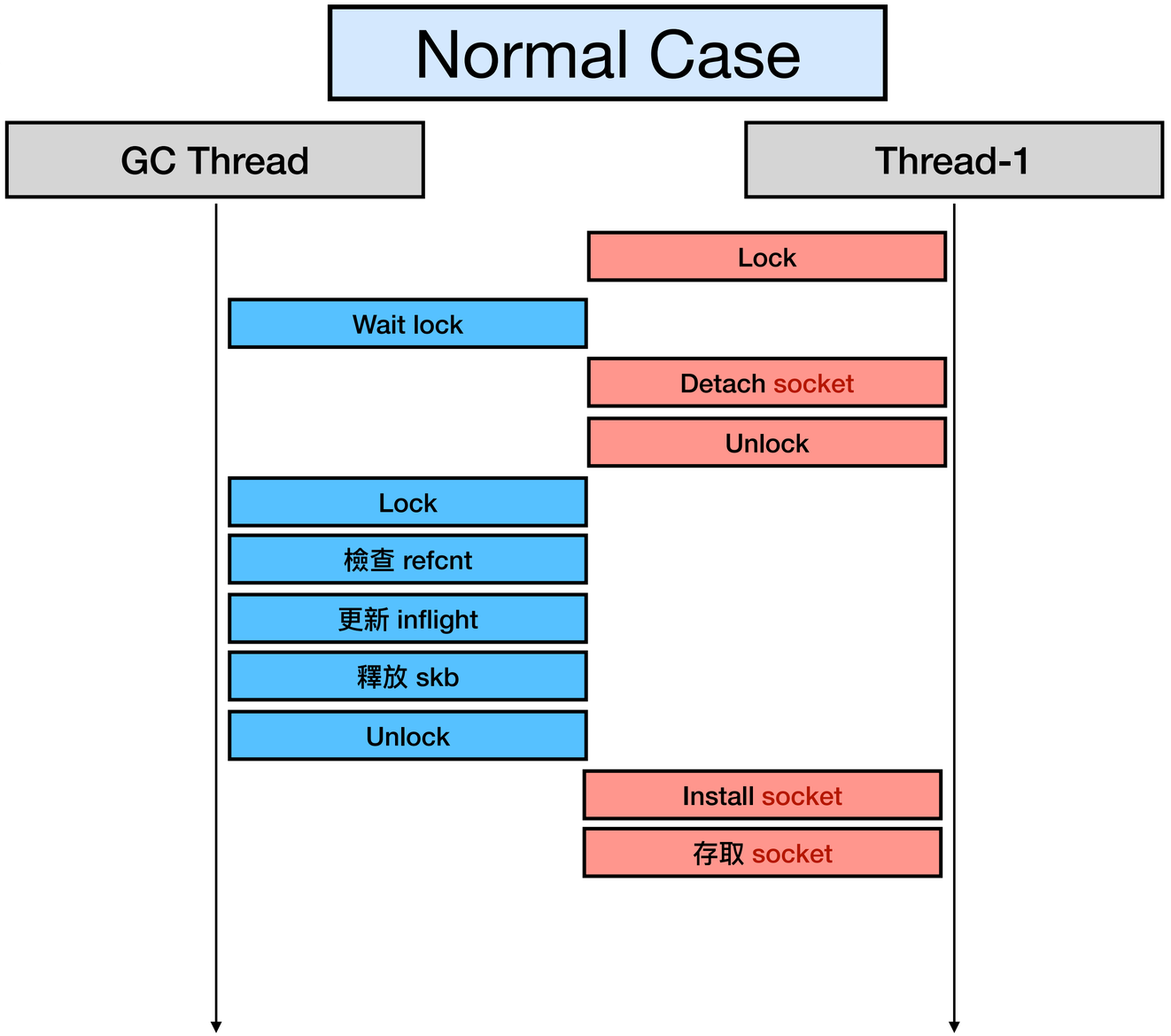
AF_UNIX

210728



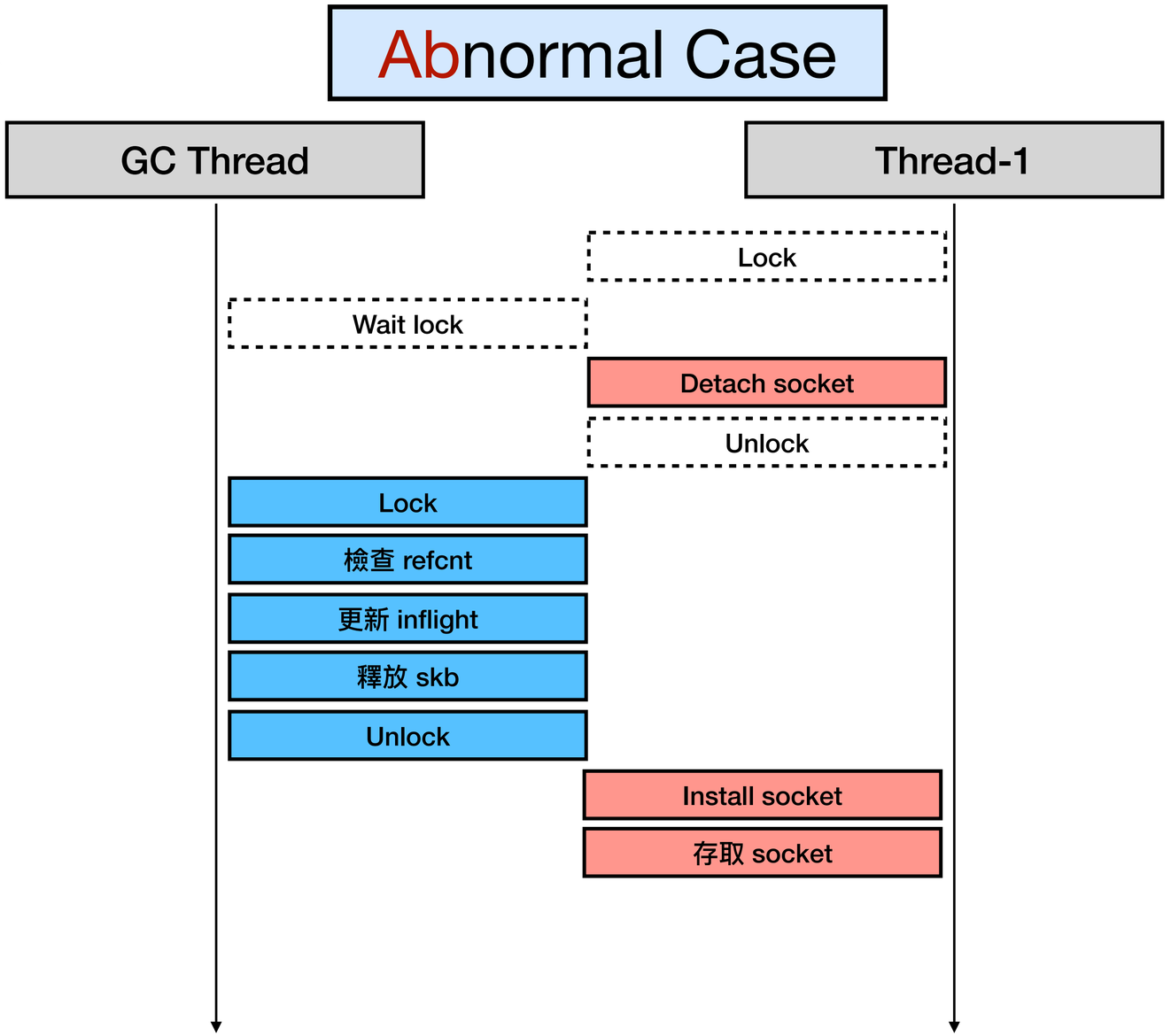
AF_UNIX

210728



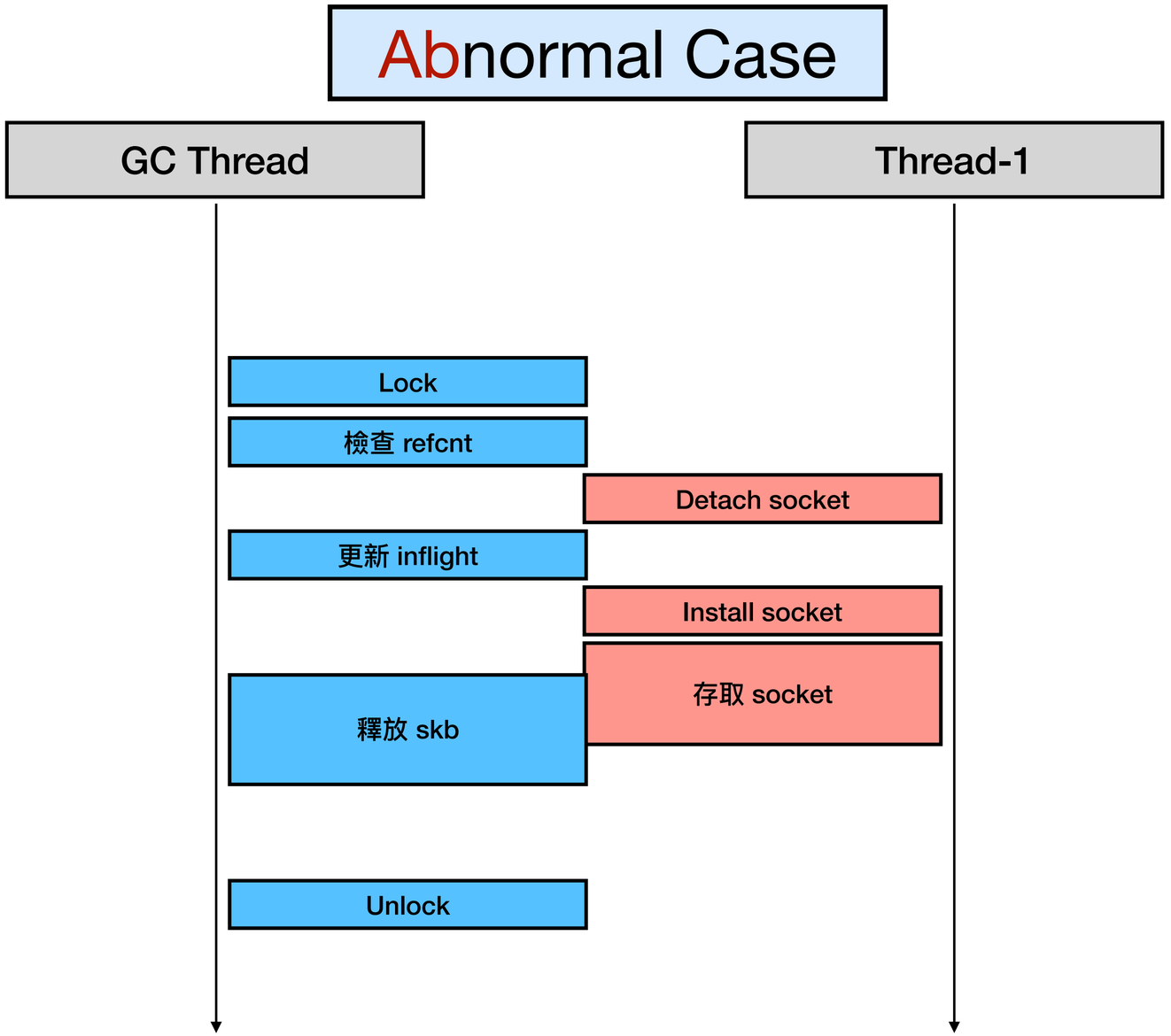
AF_UNIX

210728



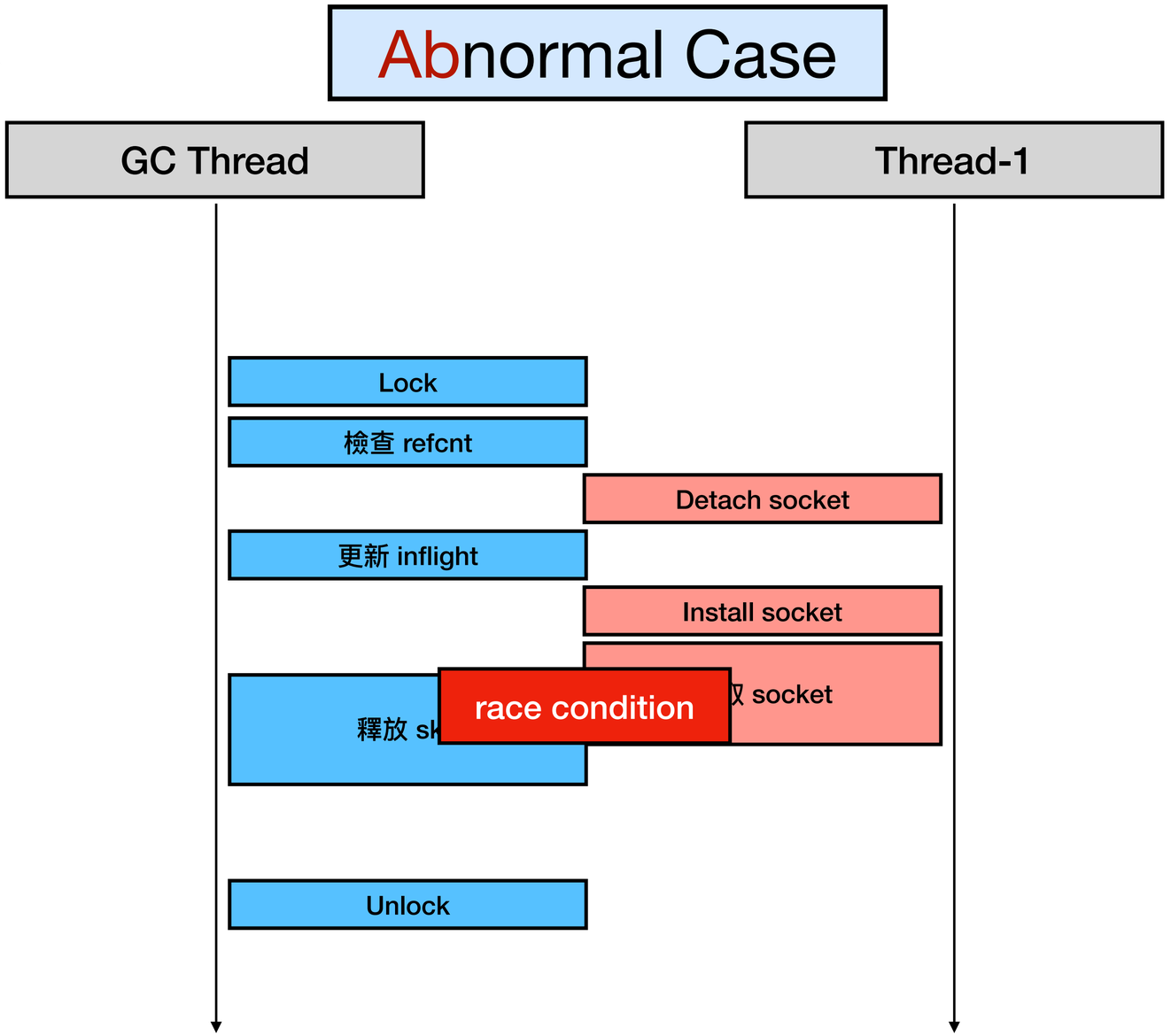
AF_UNIX

210728



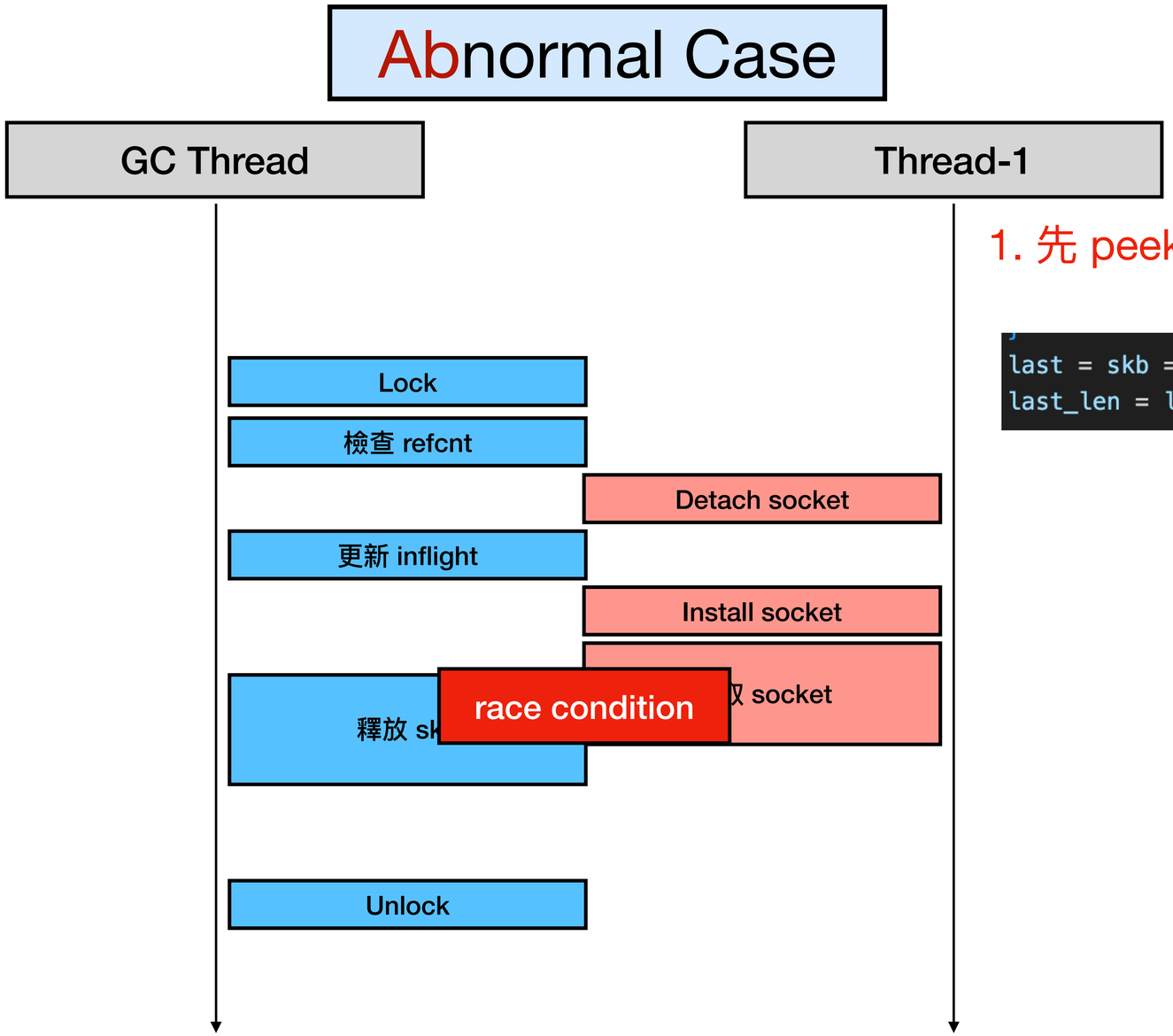
AF_UNIX

210728



AF_UNIX

210728



1. 先 peek receive queue，不會更新 socket 的 refcnt

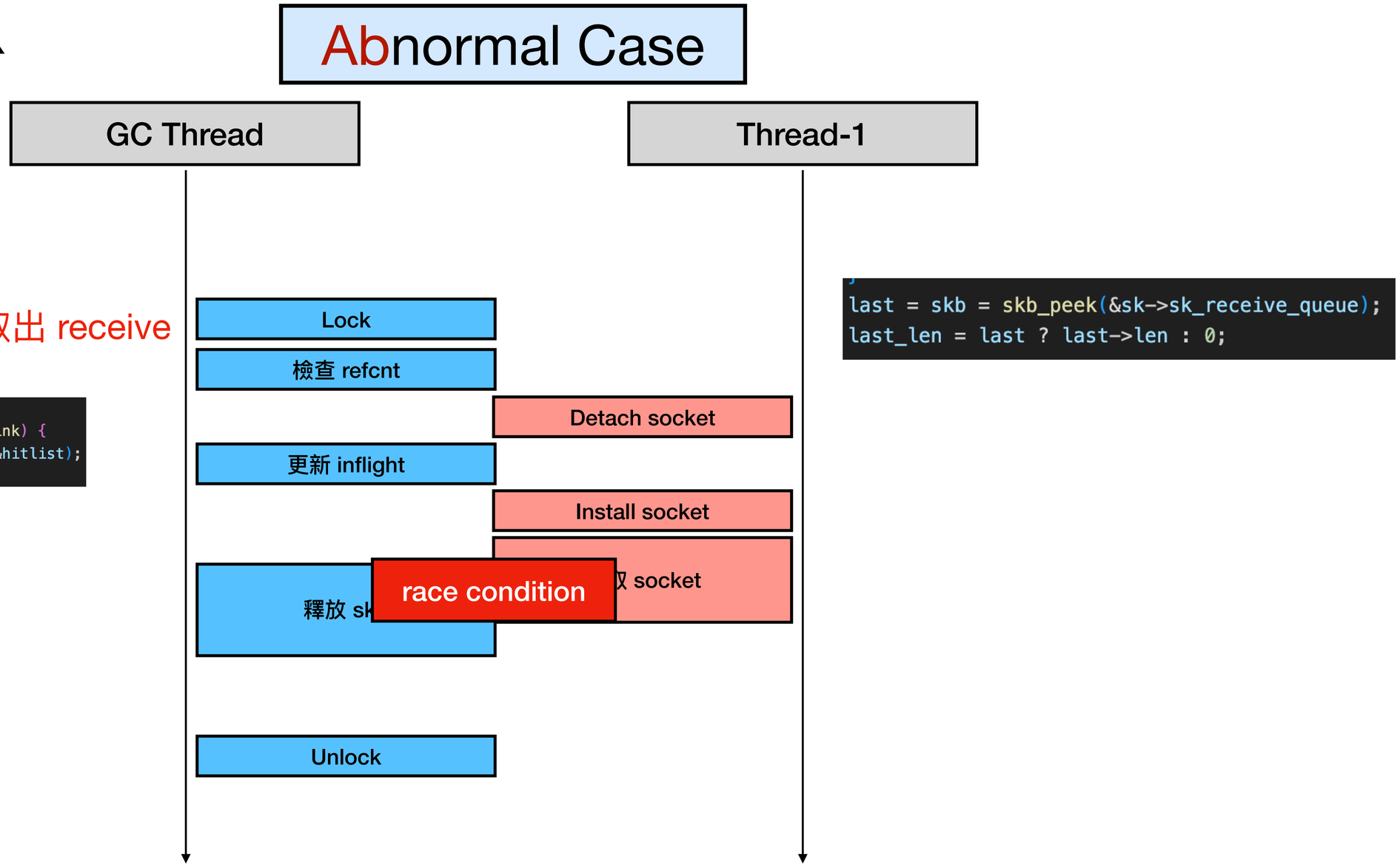
```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

AF_UNIX

210728

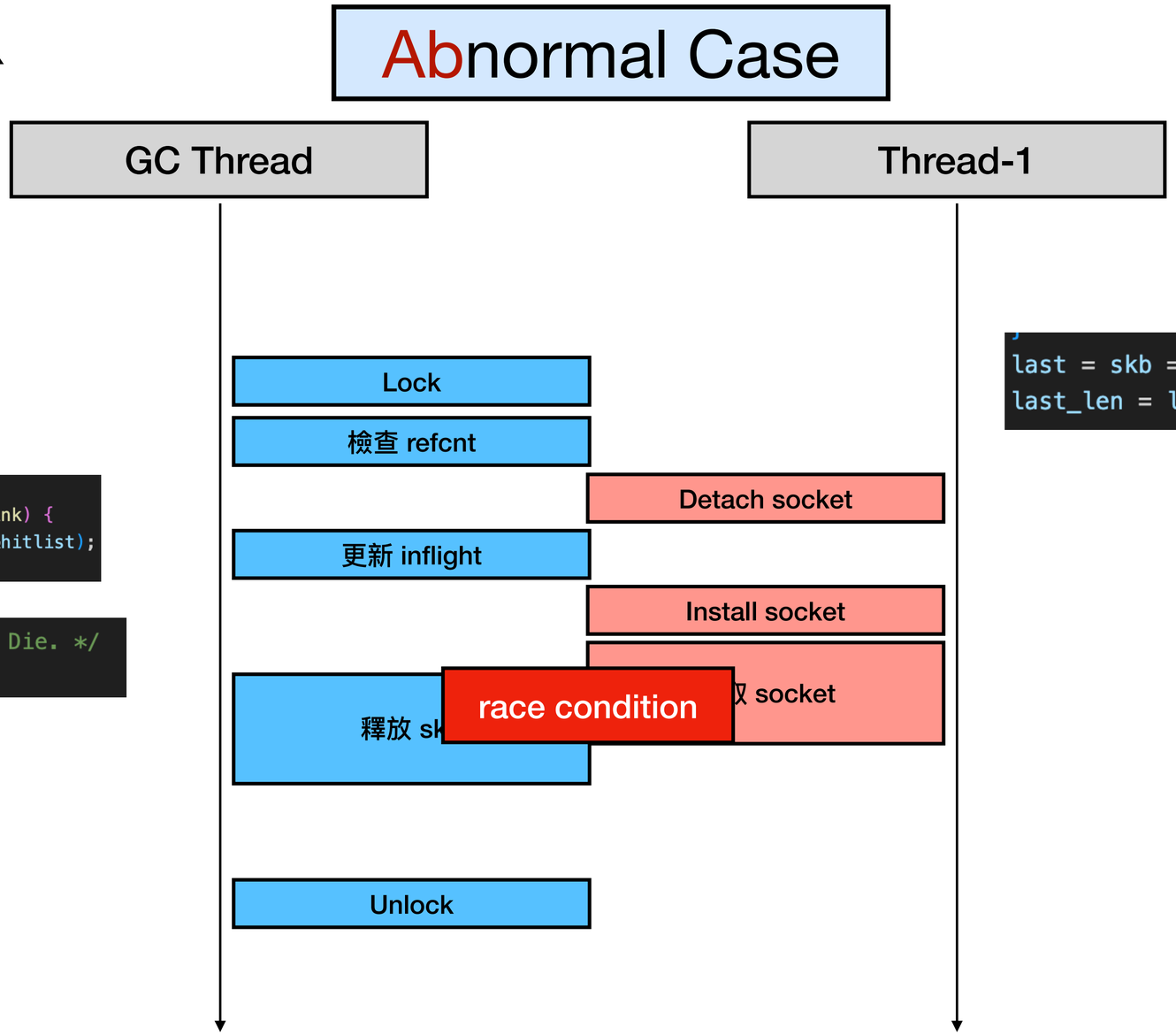
2. 處理 gc_candidates， 取出 receive queue 的 skb

```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
```



AF_UNIX

210728




```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

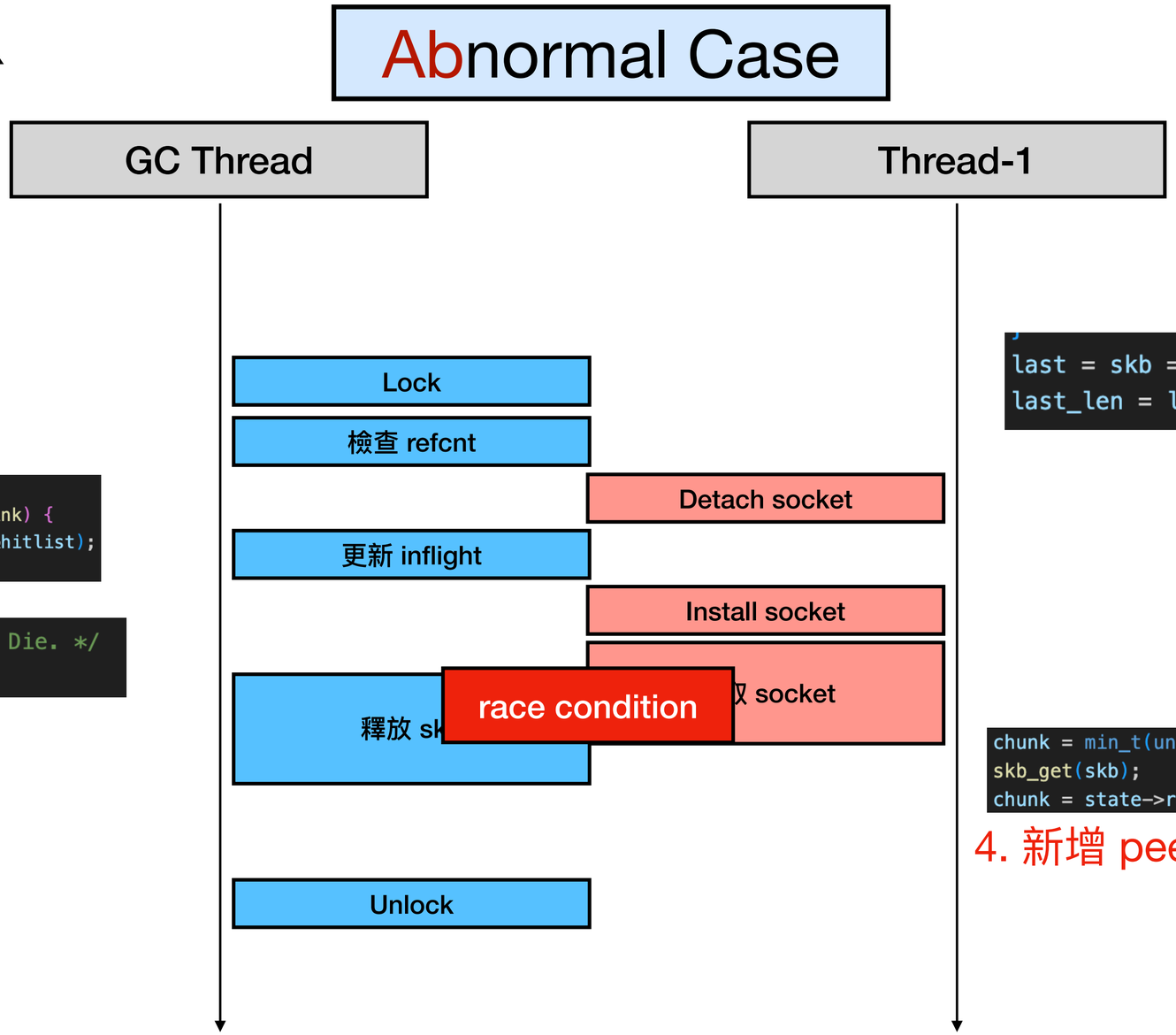
```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
```

```
/* Here we are. Hitlist is filled. Die. */
__skb_queue_purge(&hitlist);
```

3. 釋放這些 skb

AF_UNIX

210728




```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
```

```
/* Here we are. Hitlist is filled. Die. */
__skb_queue_purge(&hitlist);
```

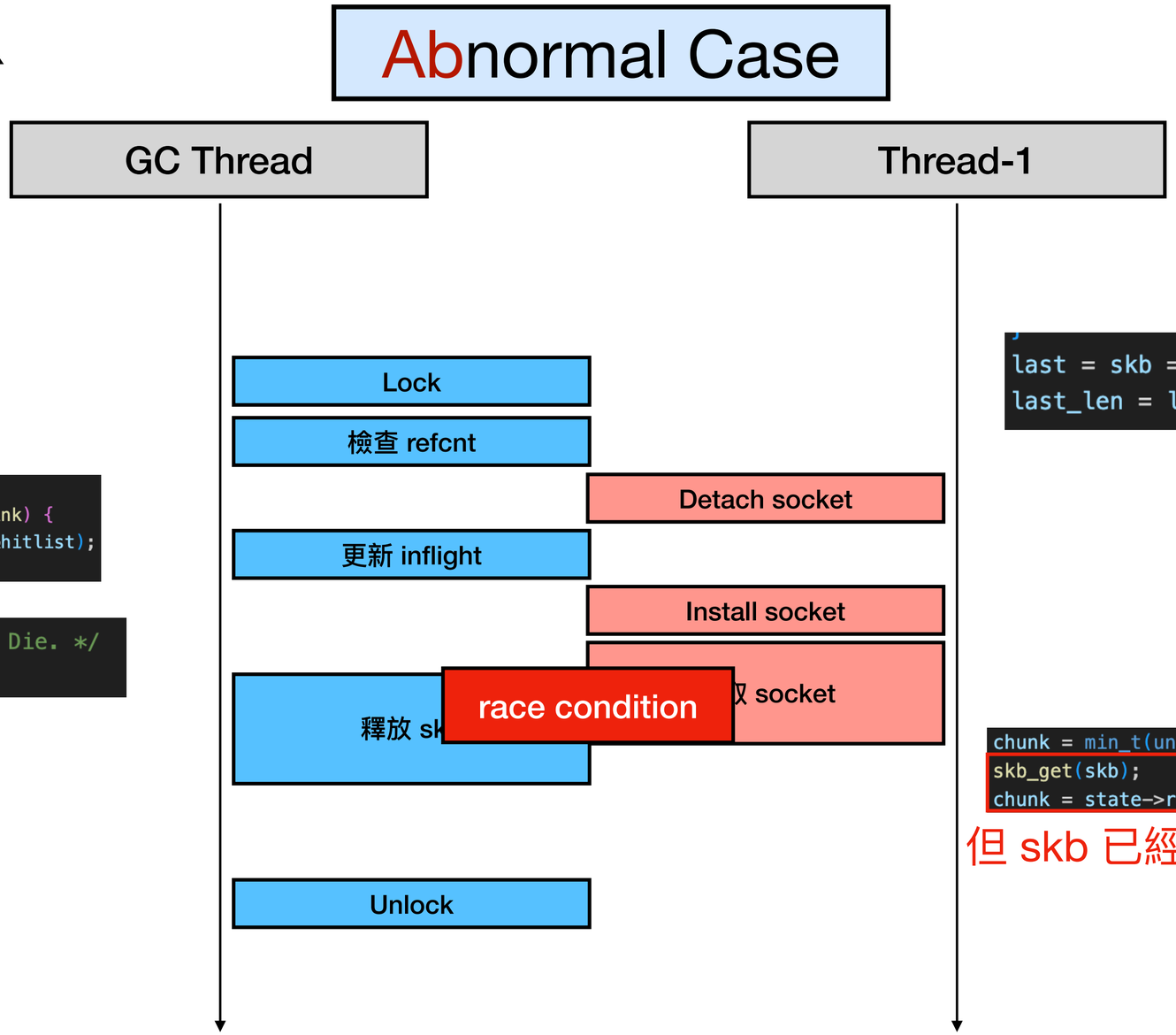
```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

```
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
```

4. 新增 peeked skb 的 refcnt 並讀資料

AF_UNIX

210728




```
skb_queue_head_init(&hitlist);
list_for_each_entry(u, &gc_candidates, link) {
    scan_children(&u->sk, inc_inflight, &hitlist);
// [...]
```

```
/* Here we are. Hitlist is filled. Die. */
__skb_queue_purge(&hitlist);
```

```
last = skb = skb_peek(&sk->sk_receive_queue);
last_len = last ? last->len : 0;
```

```
chunk = min_t(unsigned int, unix_skb_len(skb) - skip, size);
skb_get(skb);
chunk = state->recv_actor(skb, skip, chunk, state);
```

但 skb 已經被釋放掉了，因此觸發 UAF

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - 如果沒有上 lock 會發生什麼事？

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - 如果沒有上 lock 會發生什麼事？
    - 當 recvmsg 包含 MSG_PEEK flag 時，能在沒有 lock 的情況下 install socket file

```
if (!(flags & MSG_PEEK)) {…
} else {
    if (UNIXCB(skb).fp)
        scm.fp = scm_fp_dup(UNIXCB(skb).fp);
```

AF_UNIX
210728

- af_unix: fix garbage collect vs MSG_PEEK (CVE-2021-0920)
  - Patch
    - Detach 後多補上一段 lock / unlock 的操作，保證了 (?) 不會在 GC 的執行過程中 install socket
  - 體感上應該還有更好的 patch 方式

```
+       spin_lock(&unix_gc_lock);
+       spin_unlock(&unix_gc_lock);
```

```
scm->fp = scm_fp_dup(UNIXCB(skb).fp);
// [...]
spin_lock(&unix_gc_lock);
spin_unlock(&unix_gc_lock);
```

AF_UNIX
210728

- 參考資料：
  - CVE-2021-0920: Android sk_buff use-after-free in Linux
  - The quantum state of Linux kernel garbage collection CVE-2021-0920 (Part I)

AF_UNIX
211203

- fget: check that the fd still exists after getting a ref to it (CVE-2021-4083)
  - 成因與 CVE-2021-0920 類似
  - GC 透過比較 socket file 的 refcnt 與 inflight count 來保證 file object 不會被外部 fd table 存取到
  - 然而 struct file 仍可能會在 RCU read-side critical section 存取到

AF_UNIX

211203

- fget: check that the fd still exists after getting a ref to it (CVE-2021-4083)

1. 從 fd table 拿 file

```
file = files_lookup_fd_rcu(files, fd); // race window start
if (file) {
        /* File object ref couldn't be taken.
         * dup2() atomicity guarantee is the reason
         * we loop to catch the new file (or NULL pointer)
         */
        if (file->f_mode & mask)
                file = NULL;
        else if (!get_file_rcu_many(file, refs)) // race window end
                goto loop;
```

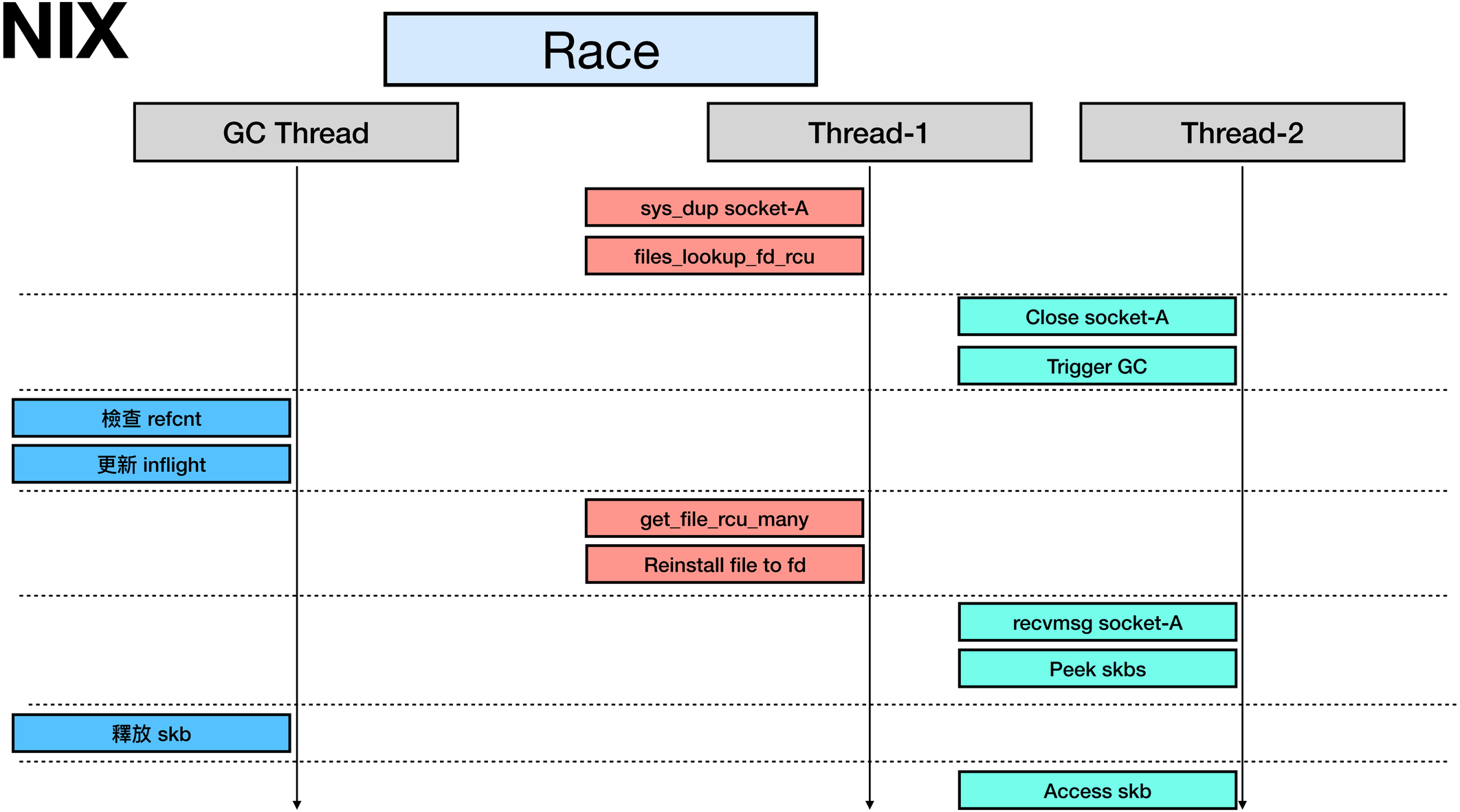
```
static inline struct file *files_lookup_fd_rcu(struct files_struct *files, unsigned int fd)
{
    RCU_LOCKDEP_WARN(!rcu_read_lock_held(),
                     "suspicious rcu_dereference_check() usage");
    return files_lookup_fd_raw(files, fd);
}
```

2. 更新 refcount

```
#define get_file_rcu_many(x, cnt)   \
    atomic_long_add_unless(&(x)->f_count, (cnt), 0)
```

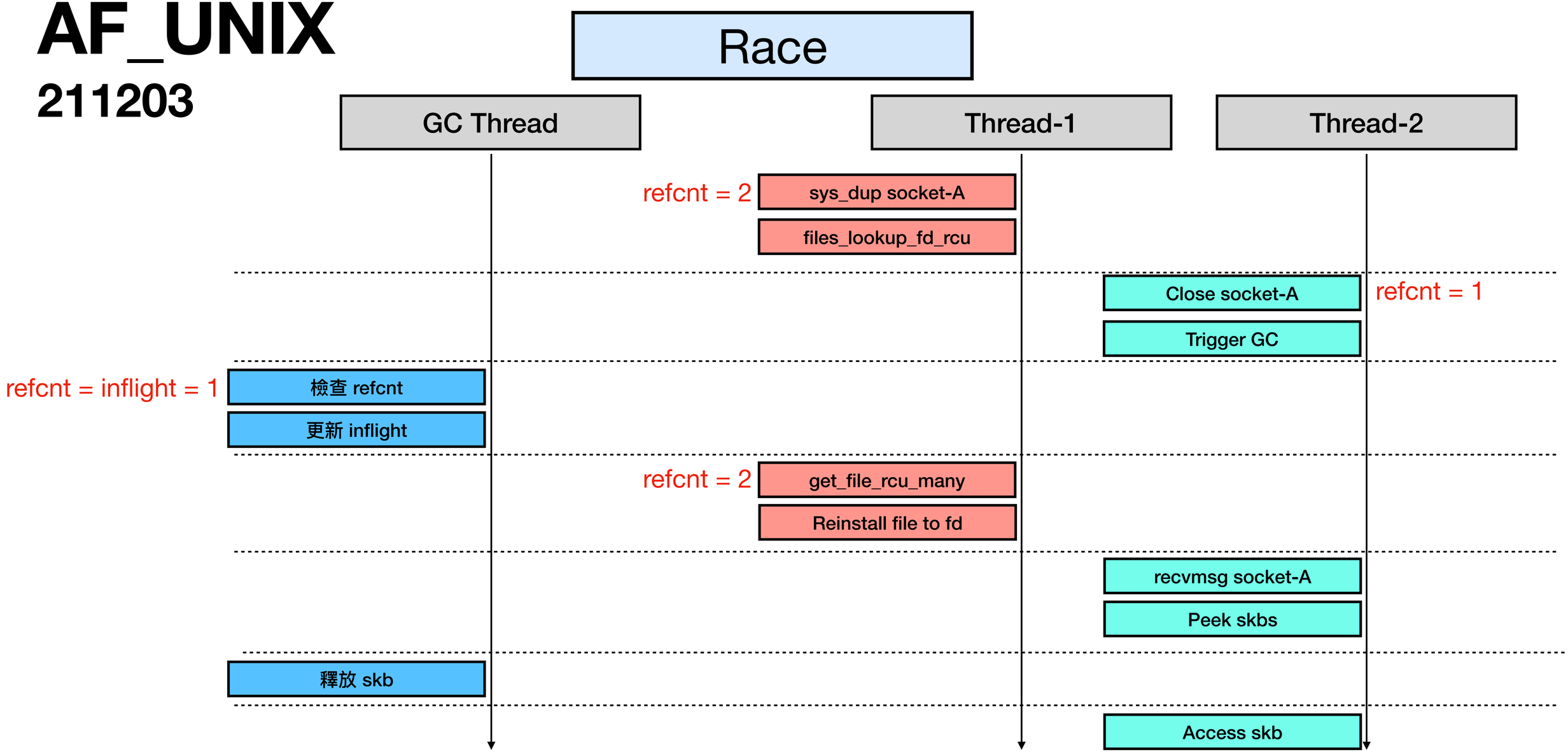
AF_UNIX

211203



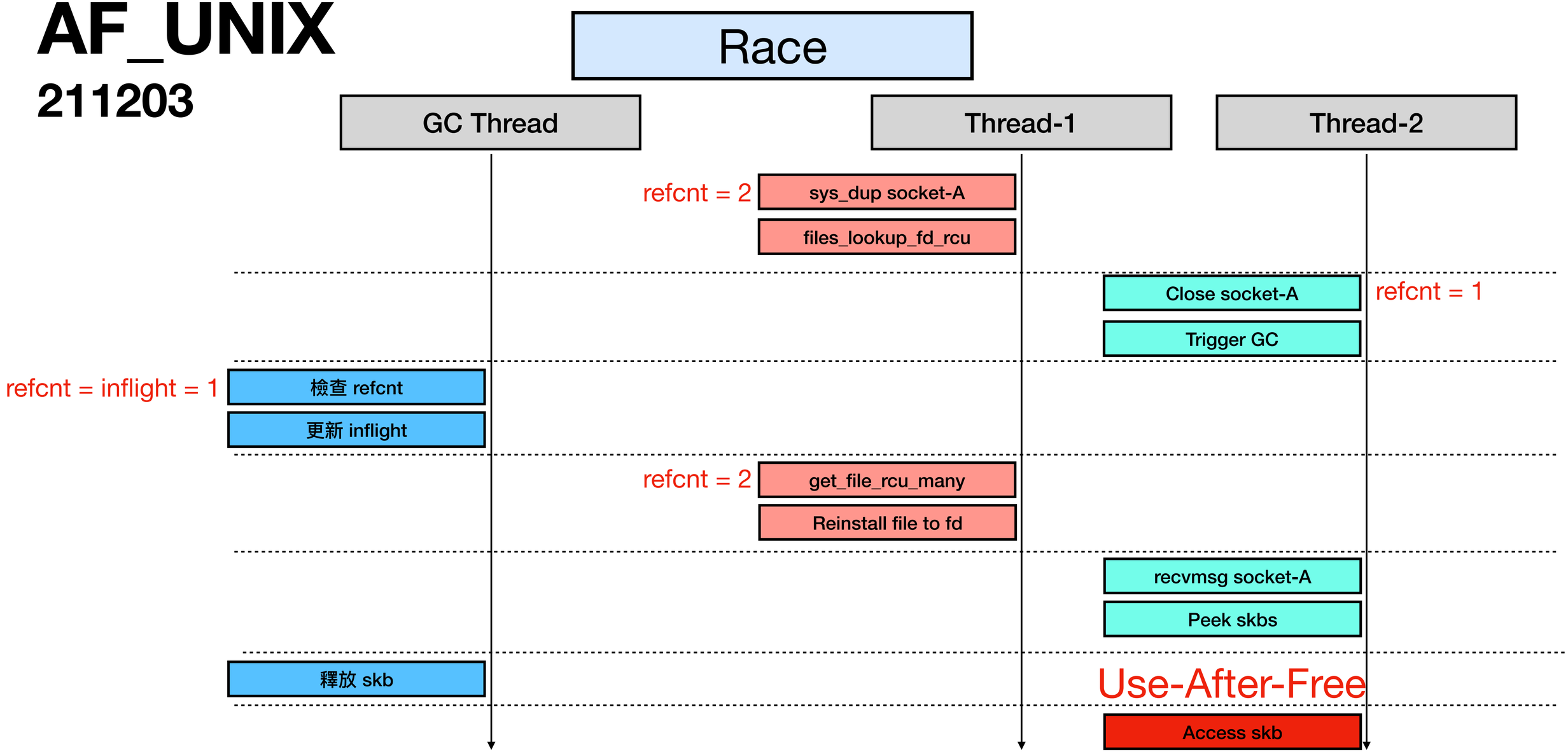
AF_UNIX

211203



AF_UNIX

211203



AF_UNIX
211203

- fget: check that the fd still exists after getting a ref to it (CVE-2021-4083)
  - Patch
    - 更新 refcnt 之後，在沒有 RCU 的保護下再存取一次 fd table
    - 重新檢查 file 是否同時也被 close
      - 如果發生的話就重新 lookup
      - 沒辦法 reinstall file

```
                        file = NULL;
                else if (!get_file_rcu_many(file, refs))
                        goto loop;
+               else if (files_lookup_fd_raw(files, fd) != file) {
+                       fput_many(file, refs);
+                       goto loop;
+               }
```

```
static inline struct file *files_lookup_fd_raw(struct files_struct *files, unsigned int fd)
{
    struct fdtable *fdt = rcu_dereference_raw(files->fdt);

    if (fd < fdt->max_fds) {
        fd = array_index_nospec(fd, fdt->max_fds);
        return rcu_dereference_raw(fdt->fd[fd]);
    }
    return NULL;
}
```

AF_UNIX

211203

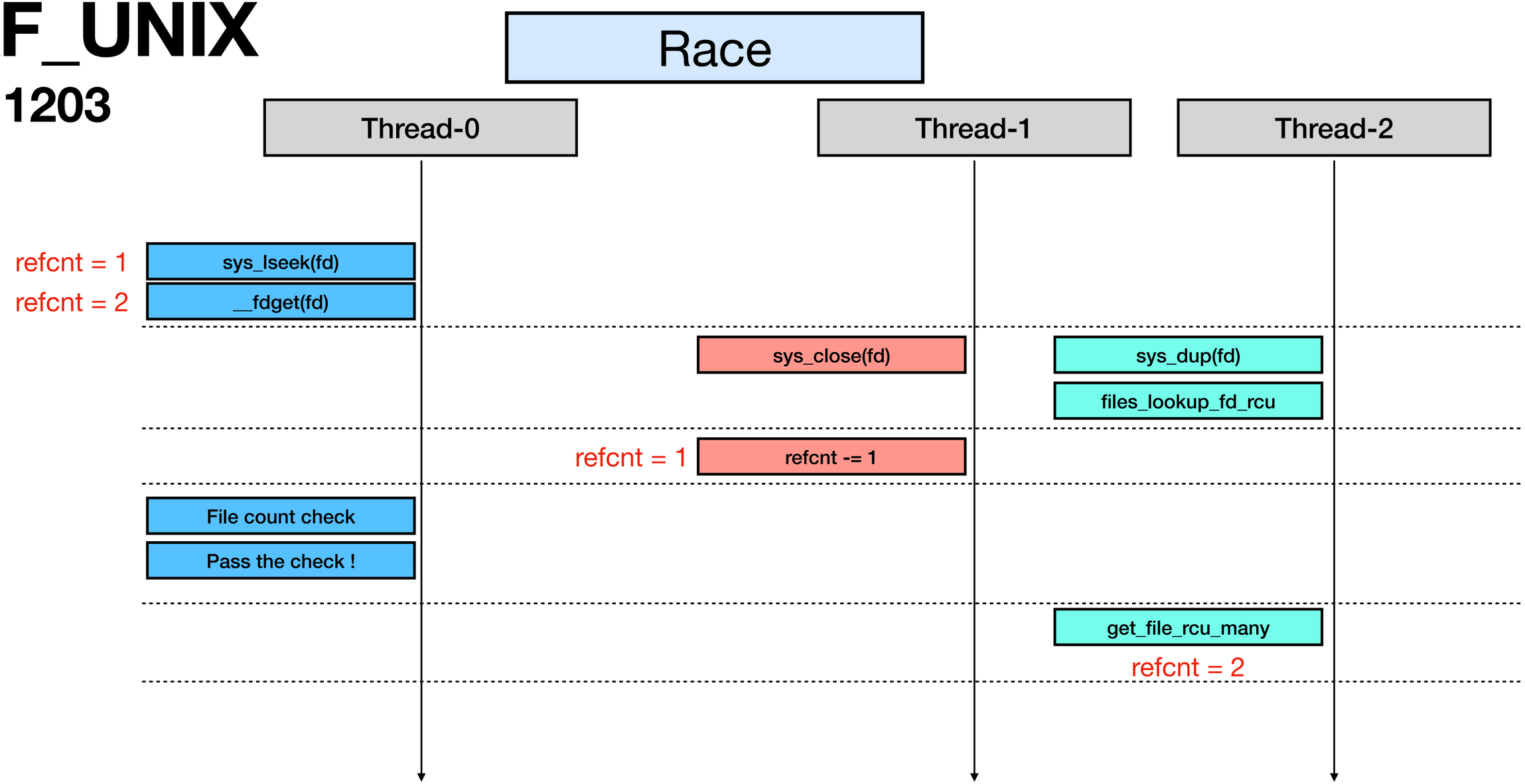
- fget: check that the fd still exists after getting a ref to it (CVE-2021-4083)
  - 其他觸發路徑 - lseek, opendir, …
    - 如果能存取多個 file object，就會上 pos_lock
    - 透過 race 可以 bypass file_count(file) 的檢查

```c
static off_t ksys_lseek(unsigned int fd, off_t offset, unsigned int whence)
{
    off_t retval;
    struct fd f = fdget_pos(fd);
```

```c
unsigned long __fdget_pos(unsigned int fd)
{
    unsigned long v = __fdget(fd);
    struct file *file = (struct file *)(v & ~3);

    if (file && (file->f_mode & FMODE_ATOMIC_POS)) {
        if (file_count(file) > 1) {
            v |= FDPUT_POS_UNLOCK;
            mutex_lock(&file->f_pos_lock);
        }
    }
    return v;
}
```

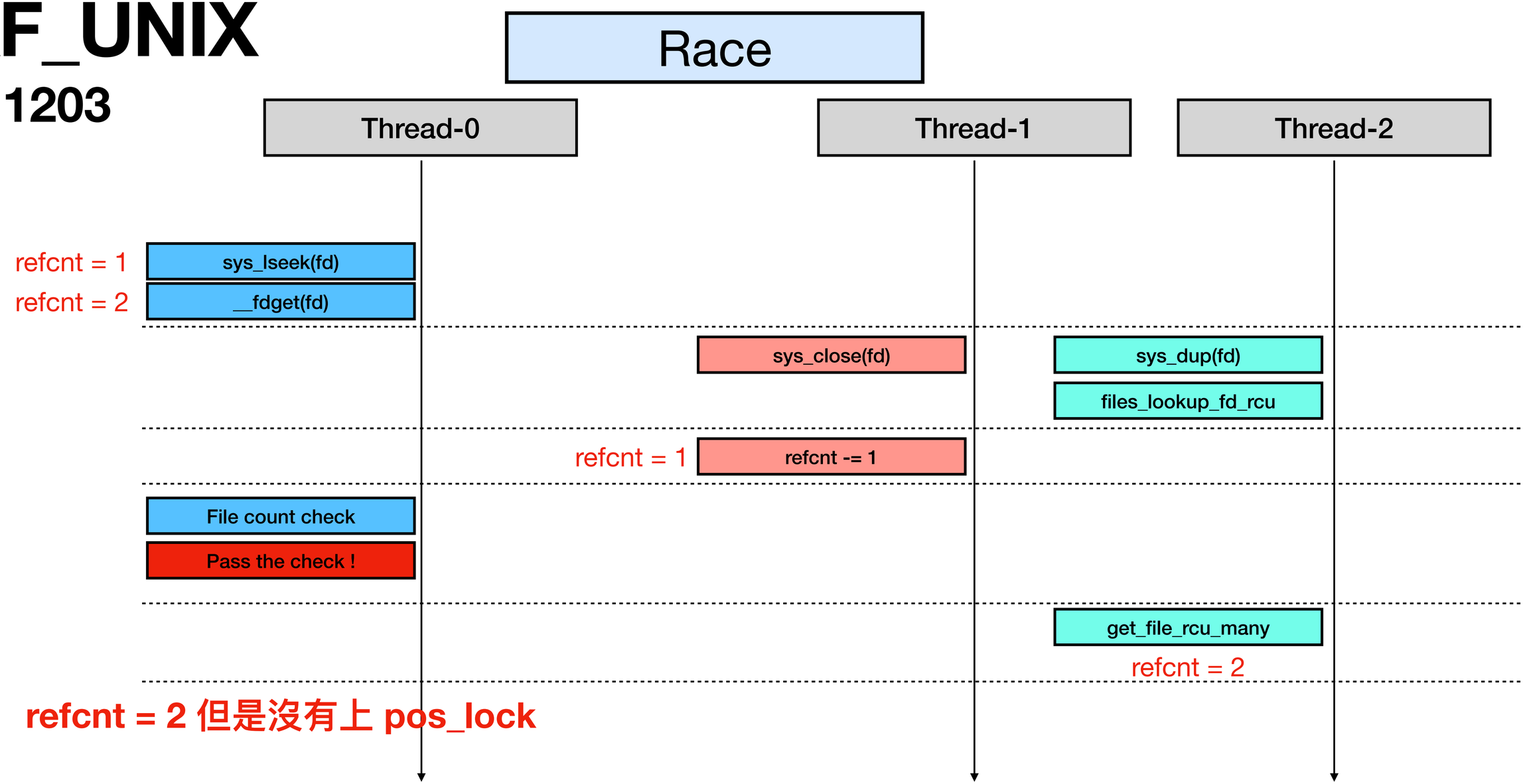
AF_UNIX

211203



AF_UNIX

211203



AF_UNIX
211203

- 參考資料：
  - Racing against the clock -- hitting a tiny kernel race window
  - Re: [fget] 054aa8d439: will-it-scale.per_thread_ops -5.7% regression

AF_UNIX

230821

- af_unix: Fix null-ptr-deref in unix_stream_sendpage() (CVE-2023-4622)
  - sys_splice from pipe to socket 時會呼叫 protocol sendpage handler
    - AF_UNIX & TCP - unix_stream_sendpage
  - 在 2023/06/20 整個 sendpage 的功能被拔掉
    - Replace 成 sendmsg(MSG_SPLICE_PAGES)

AF_UNIX

230821

```c
static int pipe_to_sendpage(struct pipe_inode_info *pipe,
                struct pipe_buffer *buf, struct splice_desc *sd)
{
    // [...]
    return file->f_op->sendpage(file, buf->page, buf->offset,
                    sd->len, &pos, more);
}
```

File sendpage op

```c
static const struct file_operations socket_file_ops = {
    .owner =    THIS_MODULE,
    // [...]
    .sendpage = sock_sendpage,
```

```c
int kernel_sendpage(struct socket *sock, struct page *page, int offset,
            size_t size, int flags)
{
    if (sock->ops->sendpage) {
        // [...]
        return sock->ops->sendpage(sock, page, offset, size, flags);
```

Type sendpage op

```c
ssize_t inet_sendpage(struct socket *sock, struct page *page, int offset,
            size_t size, int flags)
{
    struct sock *sk = sock->sk;
    if (sk->sk_prot->sendpage)
        return sk->sk_prot->sendpage(sk, page, offset, size, flags);
    return sock_no_sendpage(sock, page, offset, size, flags);
```

```c
static const struct proto_ops unix_stream_ops = {
    .family =   PF_UNIX,
    // [...]
    .sendpage = unix_stream_sendpage,
```

Protocol sendpage op

AF_UNIX

230821

- af_unix: Fix null-ptr-deref in unix_stream_sendpage() (CVE-2023-4622)
  - 假設 socket-A 與 socket-B 為 sockpair
    - [1] Thread-0 sendpage socket-A，同時 peek socket-B 的 skb
    - [2] GC-Thread 更新 inflight socket file 並釋放 socket-B 的 skbs
    - [3] Thread-0 操作 skb，但是該 skb 已經被釋放

AF_UNIX

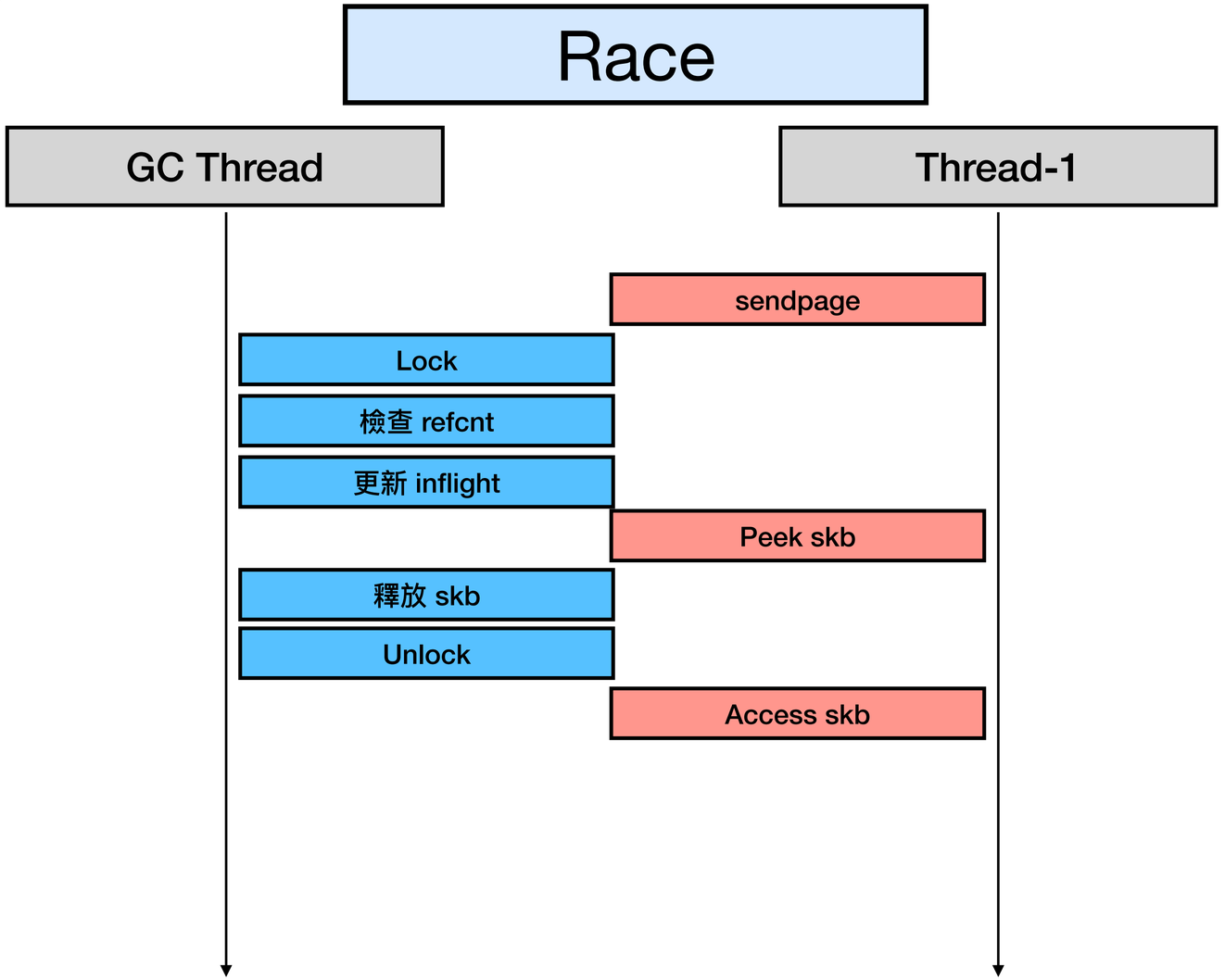
230821

- af_unix: Fix null-ptr-deref in unix_stream_sendpage() (CVE-2023-4622)
  - 假設 socket-A 與 socket-B 為 sockpair
    - [1] Thread-0 sendpage socket-A，同時 peek socket-B 的 skb
    - [2] GC-Thread 更新 inflight socke
    - [3] Thread-0 操作 skb，但是該 skb 已經被釋放

Lockless !

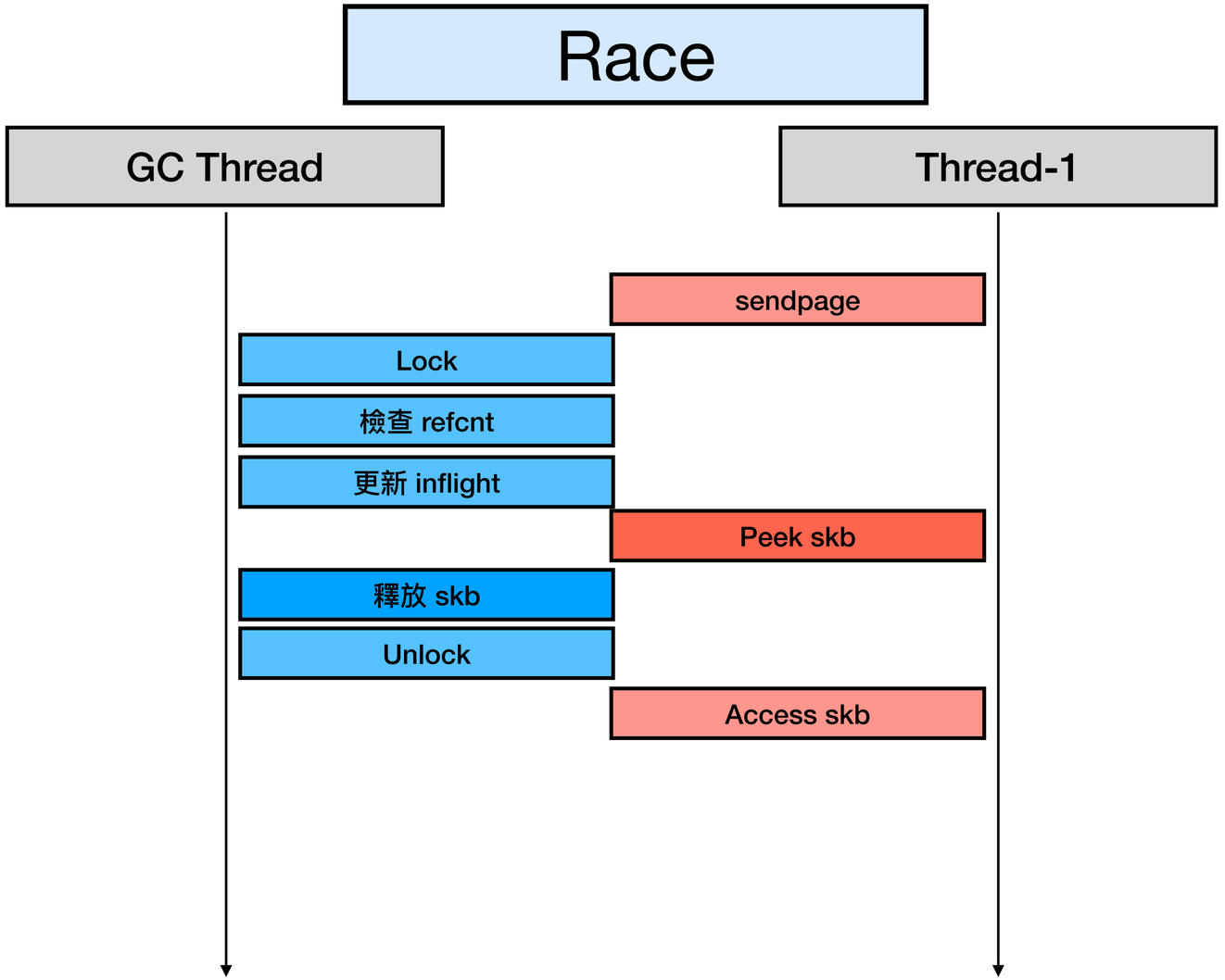
AF_UNIX

230821



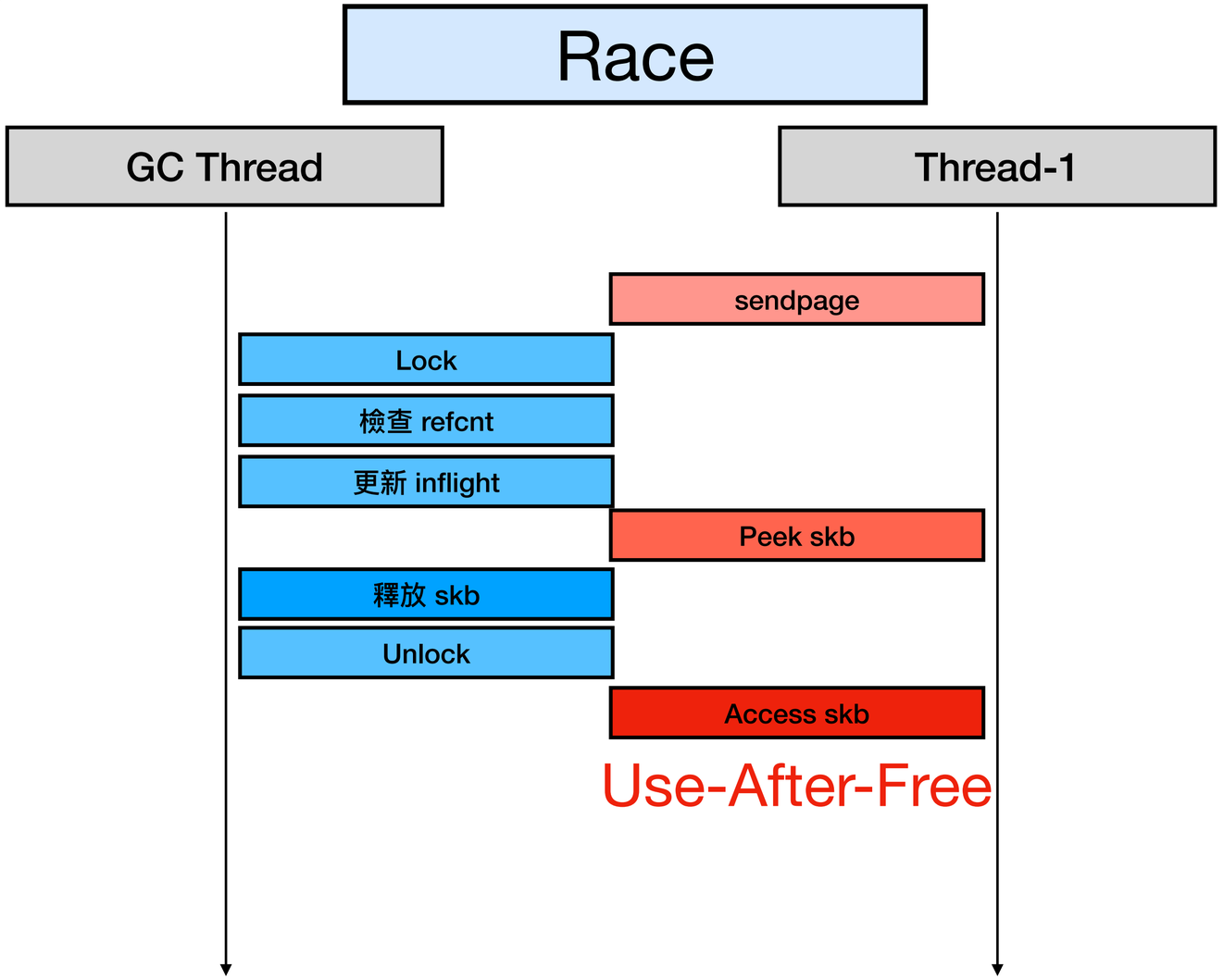
AF_UNIX

230821



AF_UNIX

230821



AF_UNIX

230821

- af_unix: Fix null-ptr-deref in unix_stream_sendpage() (CVE-2023-4622)
  - Patch
    - [1] 用 receive_queue lock 保護 peek 操作
    - [2] 這樣能夠確保 skb 不會被放到 hitlist 當中

```
+       spin_lock(&other->sk_receive_queue.lock);
        skb = skb_peek_tail(&other->sk_receive_queue);
        if (tail && tail == skb) {
                skb = newskb;
@@ -2359,14 +2361,11 @@
        refcount_add(size, &sk->sk_wmem_alloc);

        if (newskb) {
-               err = unix_scm_to_skb(&scm, skb, false);
-               if (err)
-                       goto err_state_unlock;
-               spin_lock(&other->sk_receive_queue.lock);
+               unix_scm_to_skb(&scm, skb, false);
                __skb_queue_tail(&other->sk_receive_queue, newskb);
-               spin_unlock(&other->sk_receive_queue.lock);
        }

+       spin_unlock(&other->sk_receive_queue.lock);
```

[1]

```
spin_lock(&x->sk_receive_queue.lock);
skb_queue_walk_safe(&x->sk_receive_queue, skb, next) {
    // [...]
    if (hit && hitlist != NULL) {
        __skb_unlink(skb, &x->sk_receive_queue);
        __skb_queue_tail(hitlist, skb);
    }
}
spin_unlock(&x->sk_receive_queue.lock);
```

[2]

AF_UNIX
240409

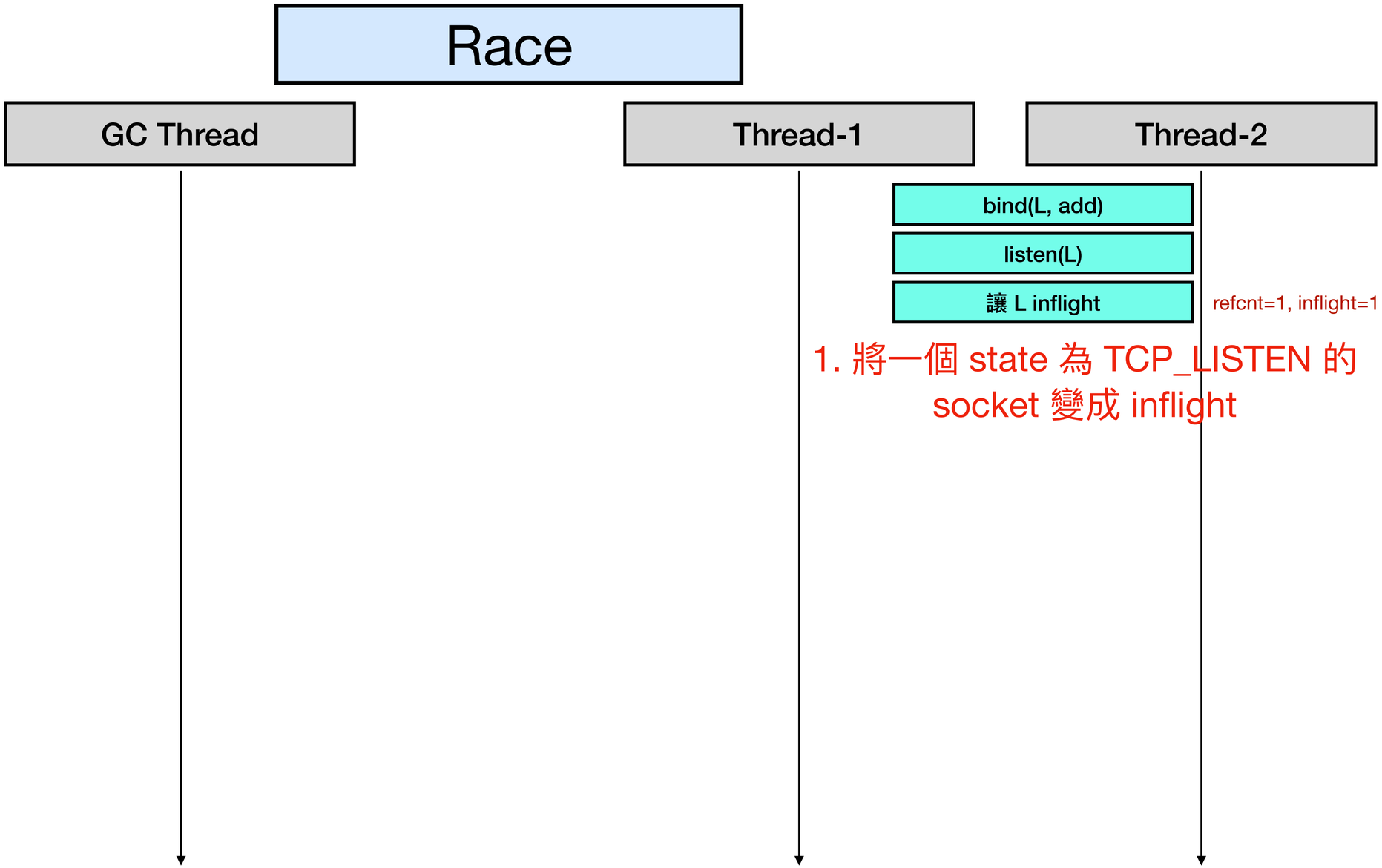
- af_unix: Fix garbage collector racing against connect() (CVE-2024-26923)
  - GC 沒有考慮到 embryo 也會 enqueue skb，因此在更新 inflight socket 時沒有上 lock
    - Embryo - 連線成功但是還沒被 accept 的 socket
  - 結果
    - Dangling pointer within the gc_inflight_list

AF_UNIX
240409

- af_unix: Fix garbage collector racing against connect() (CVE-2024-26923)
  - Commit 有詳細的流程說明
    - S - Unconnected socket
    - L - Listening socket
    - V - victim socket

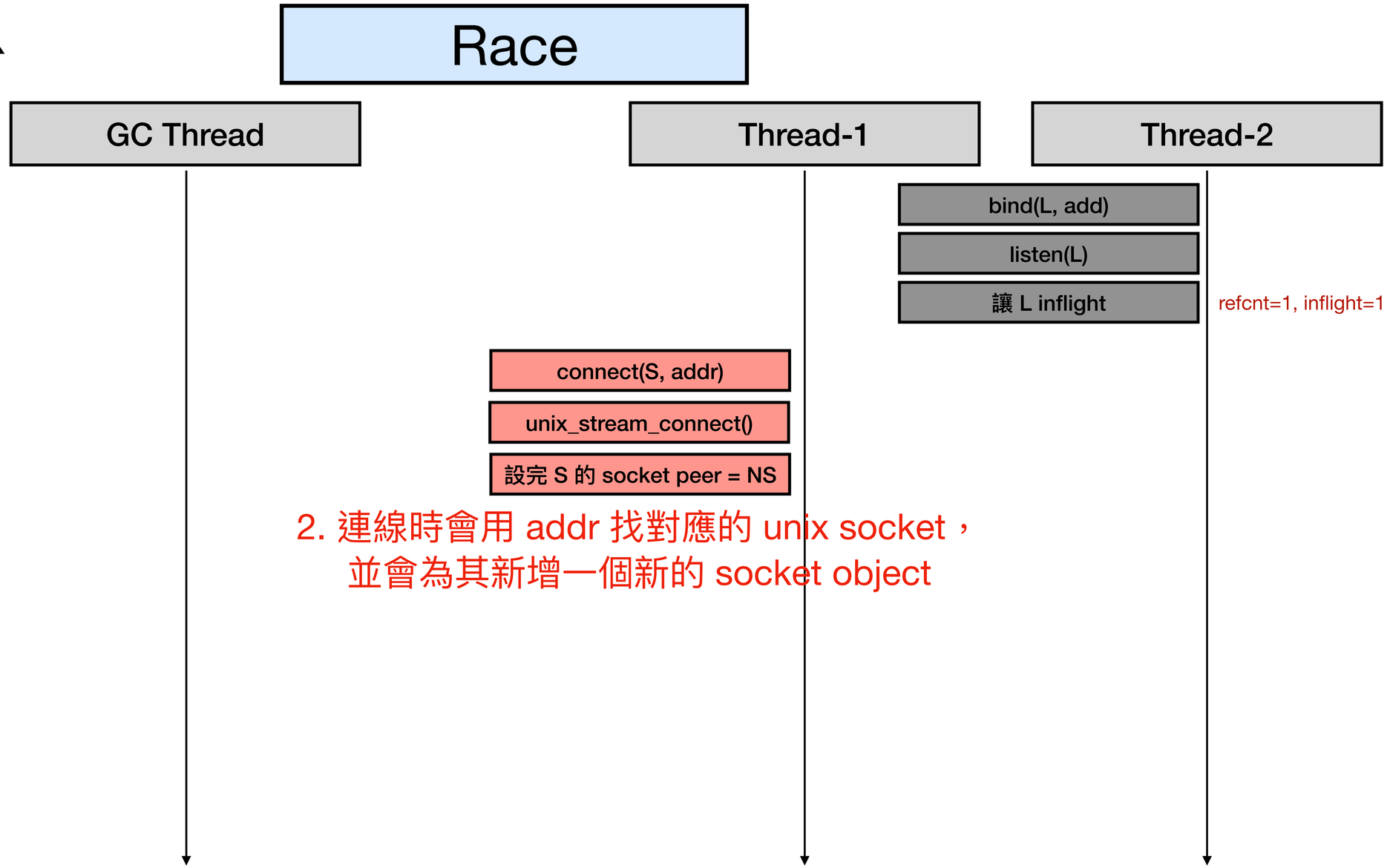
AF_UNIX

240409



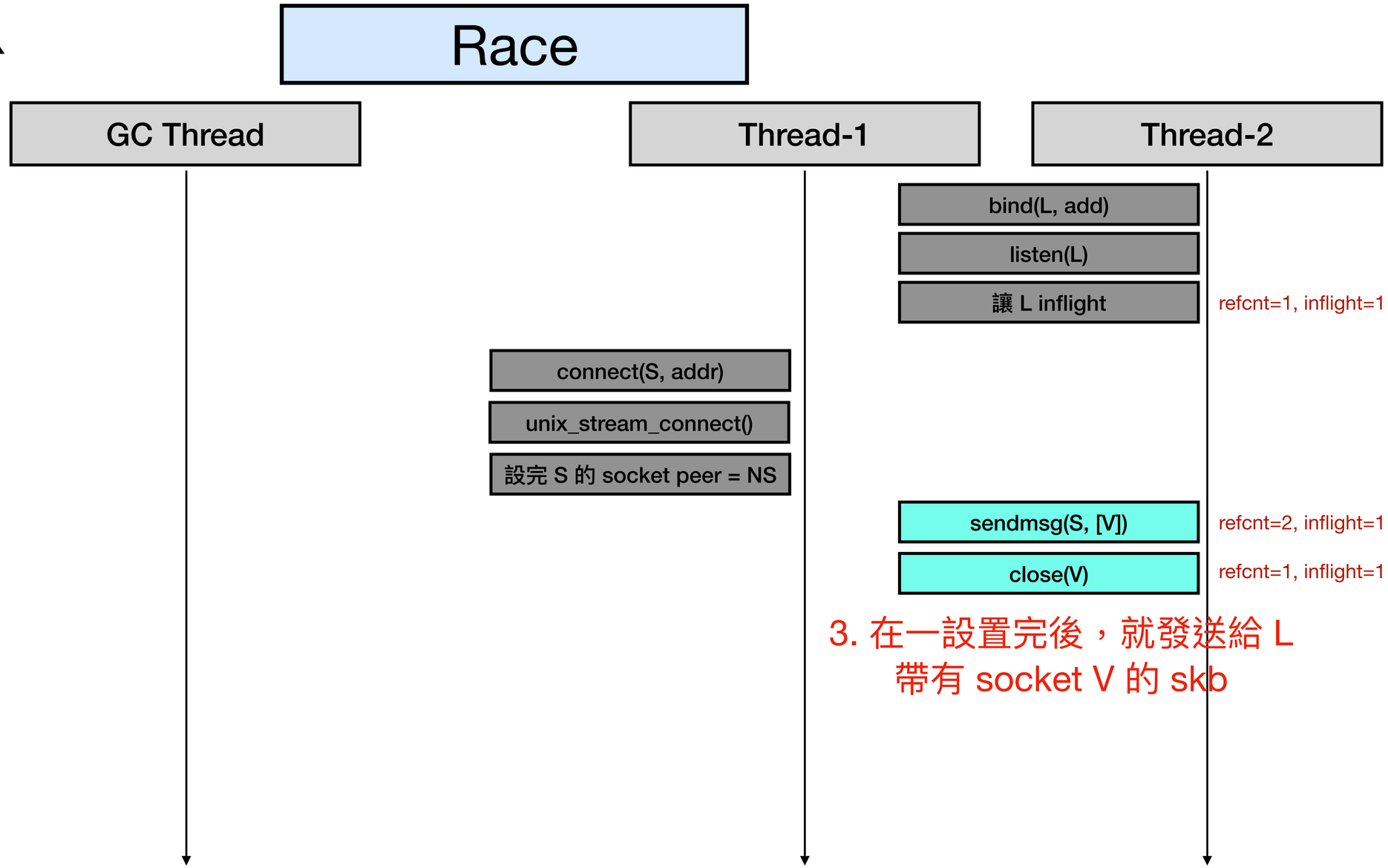
AF_UNIX

240409



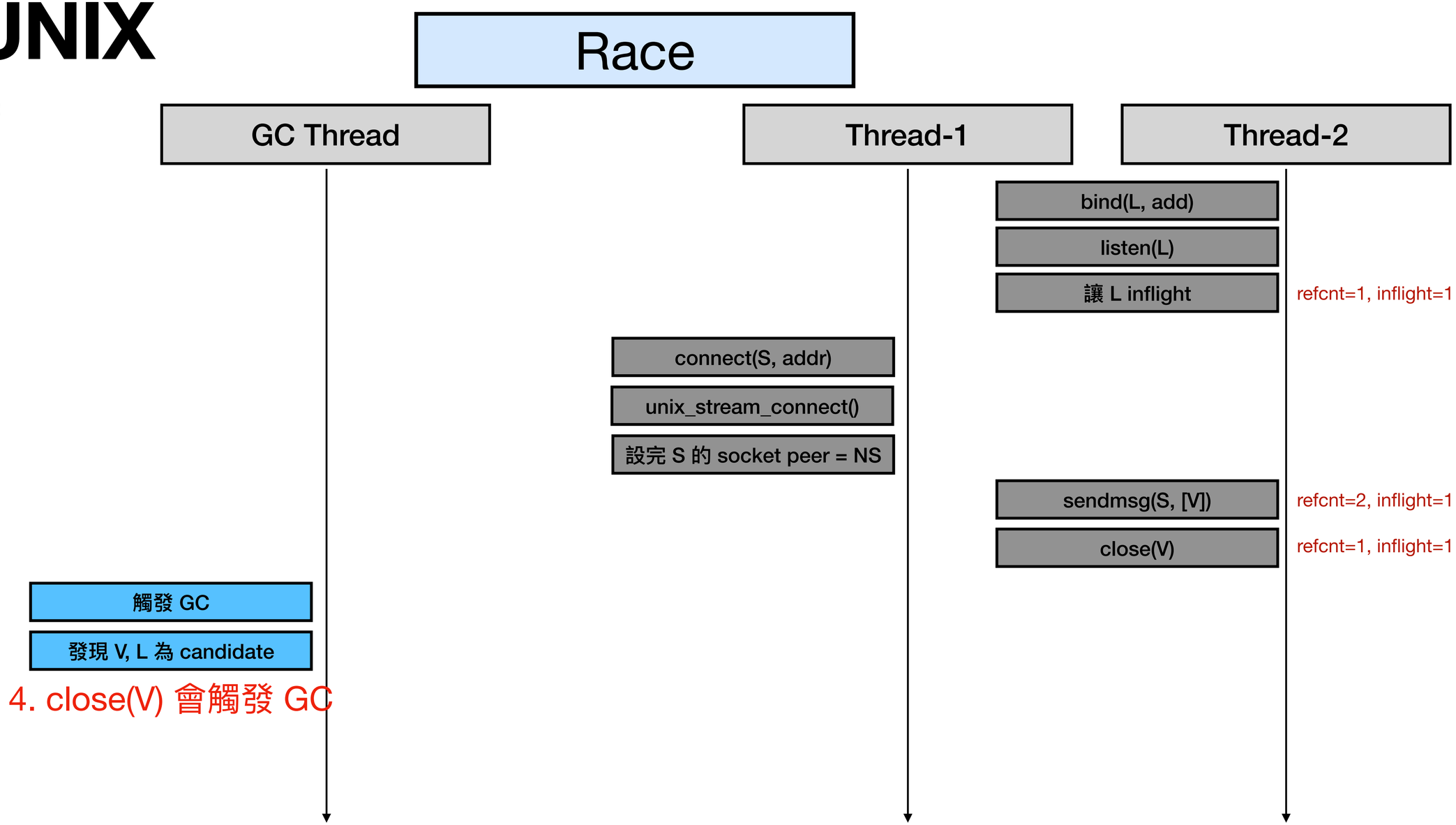
AF_UNIX

240409



AF_UNIX

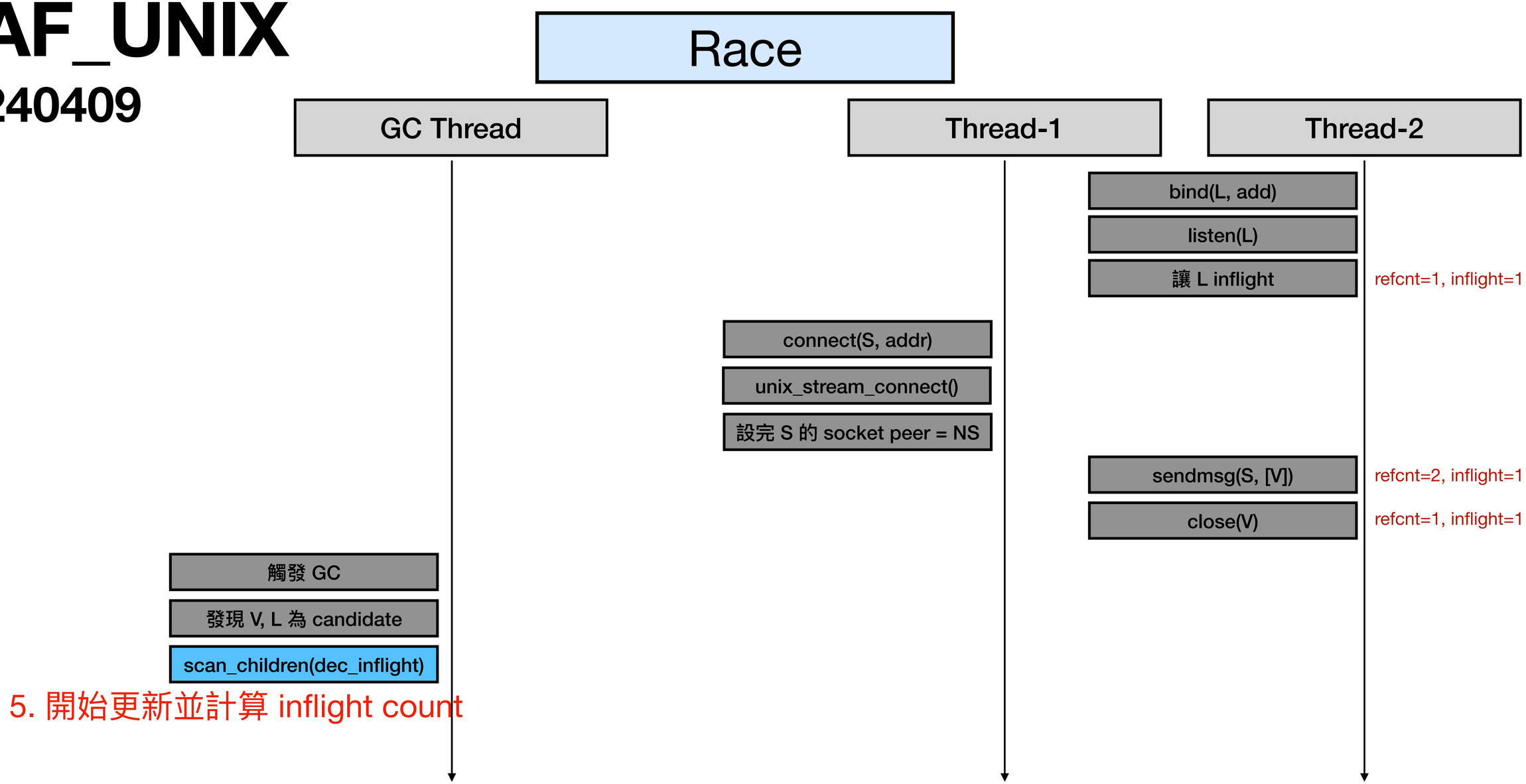
240409



4. close(V) 會觸發 GC

AF_UNIX

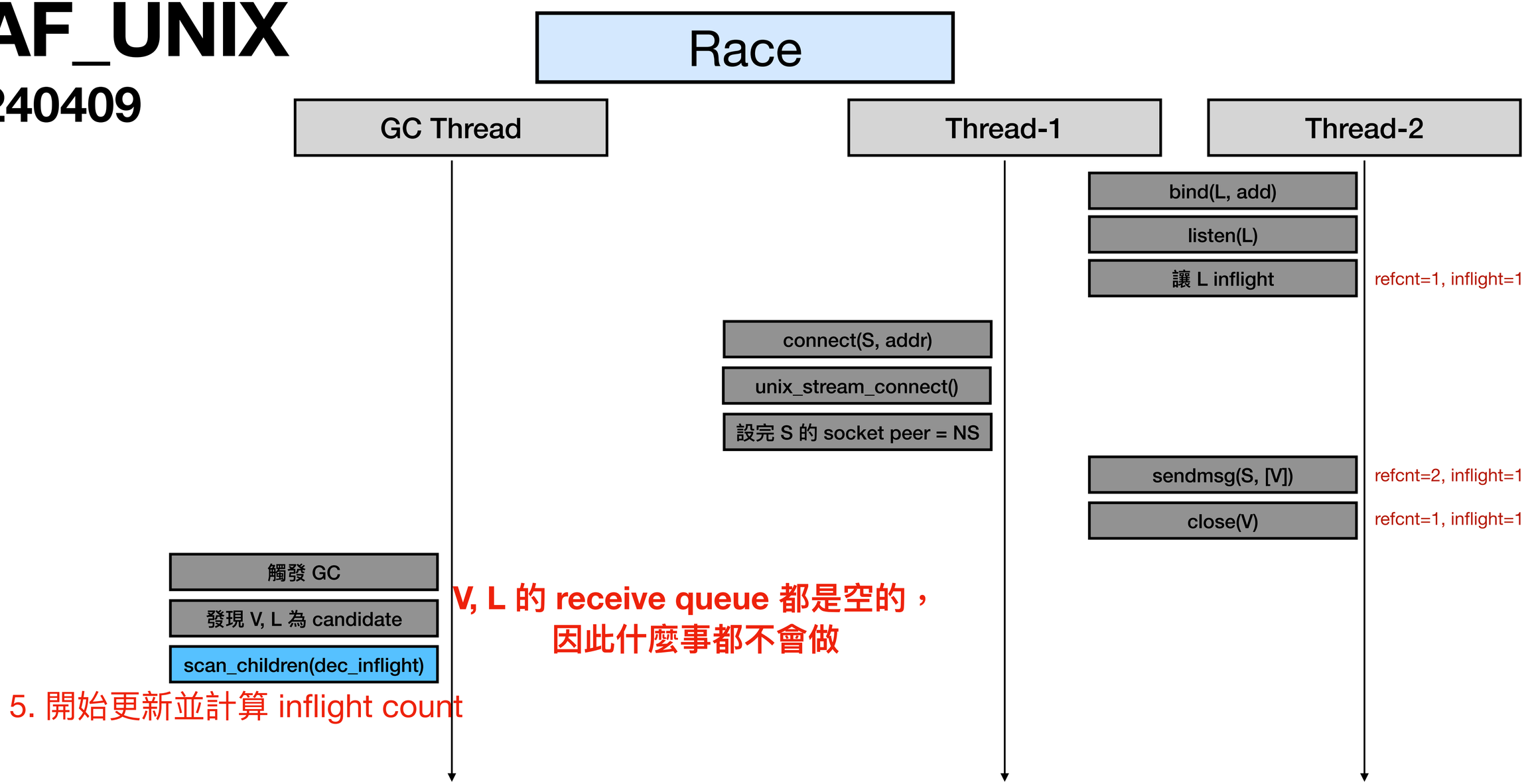
240409



5. 開始更新並計算 inflight count

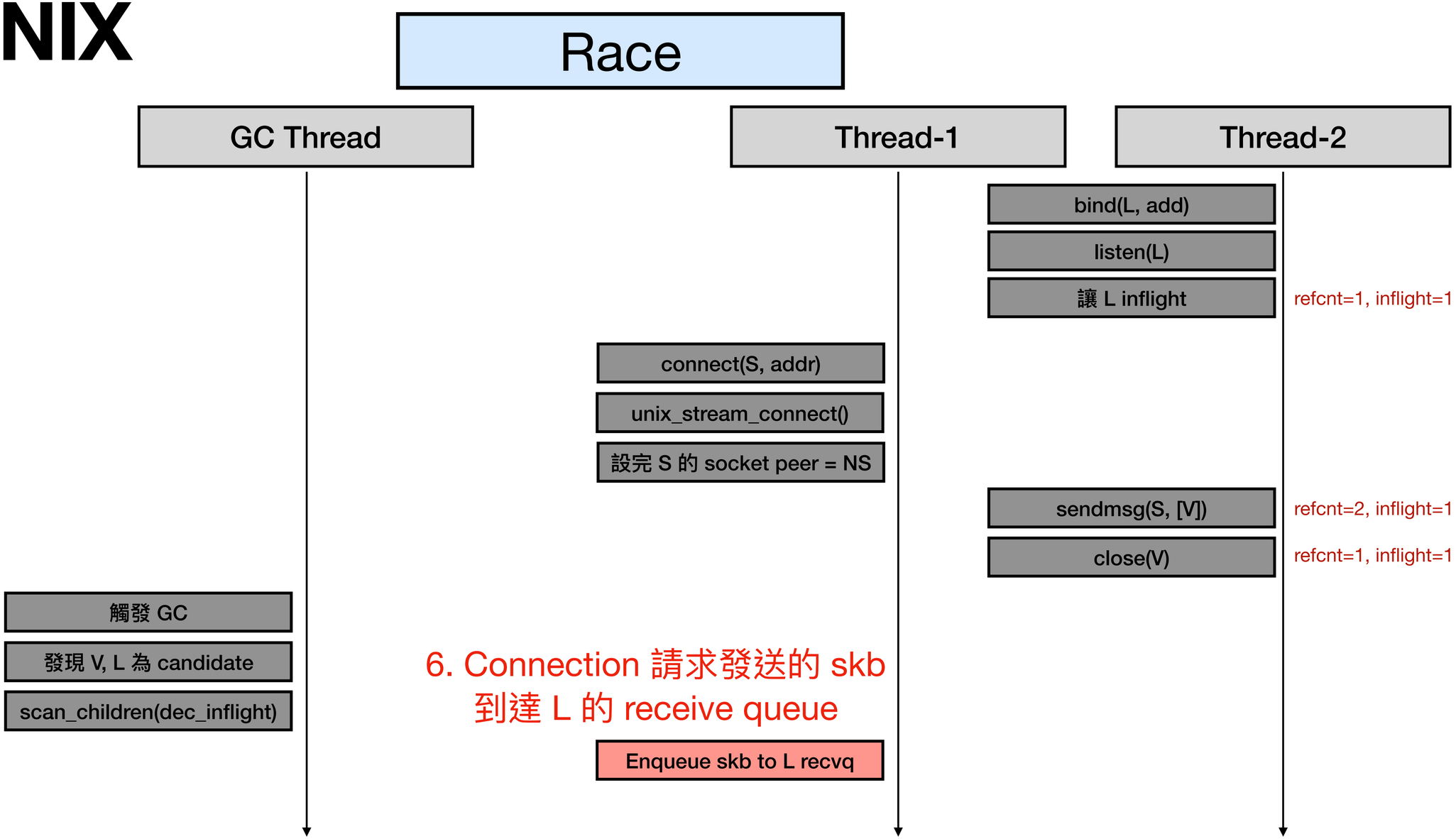
AF_UNIX

240409



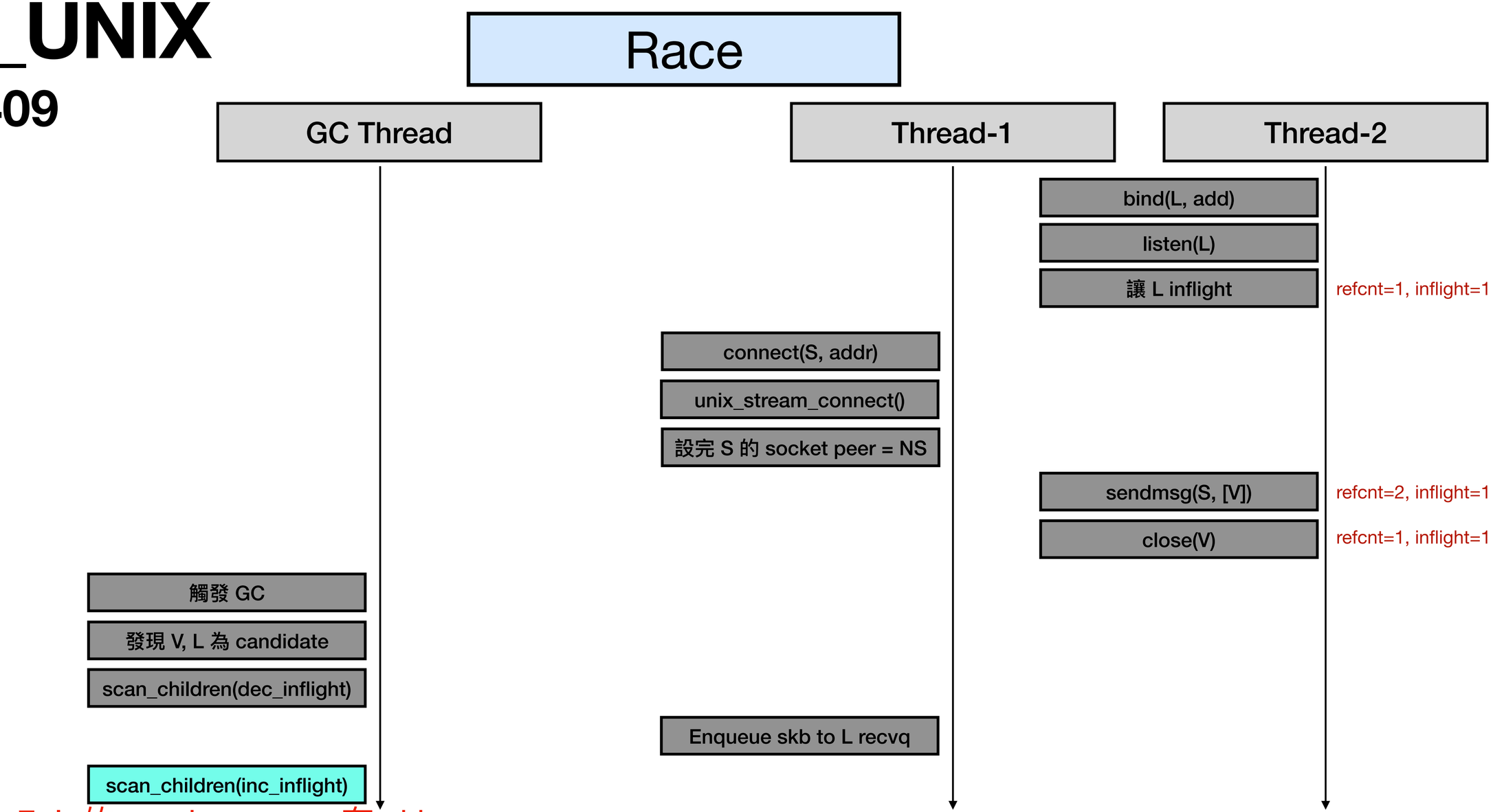
AF_UNIX

240409



AF_UNIX

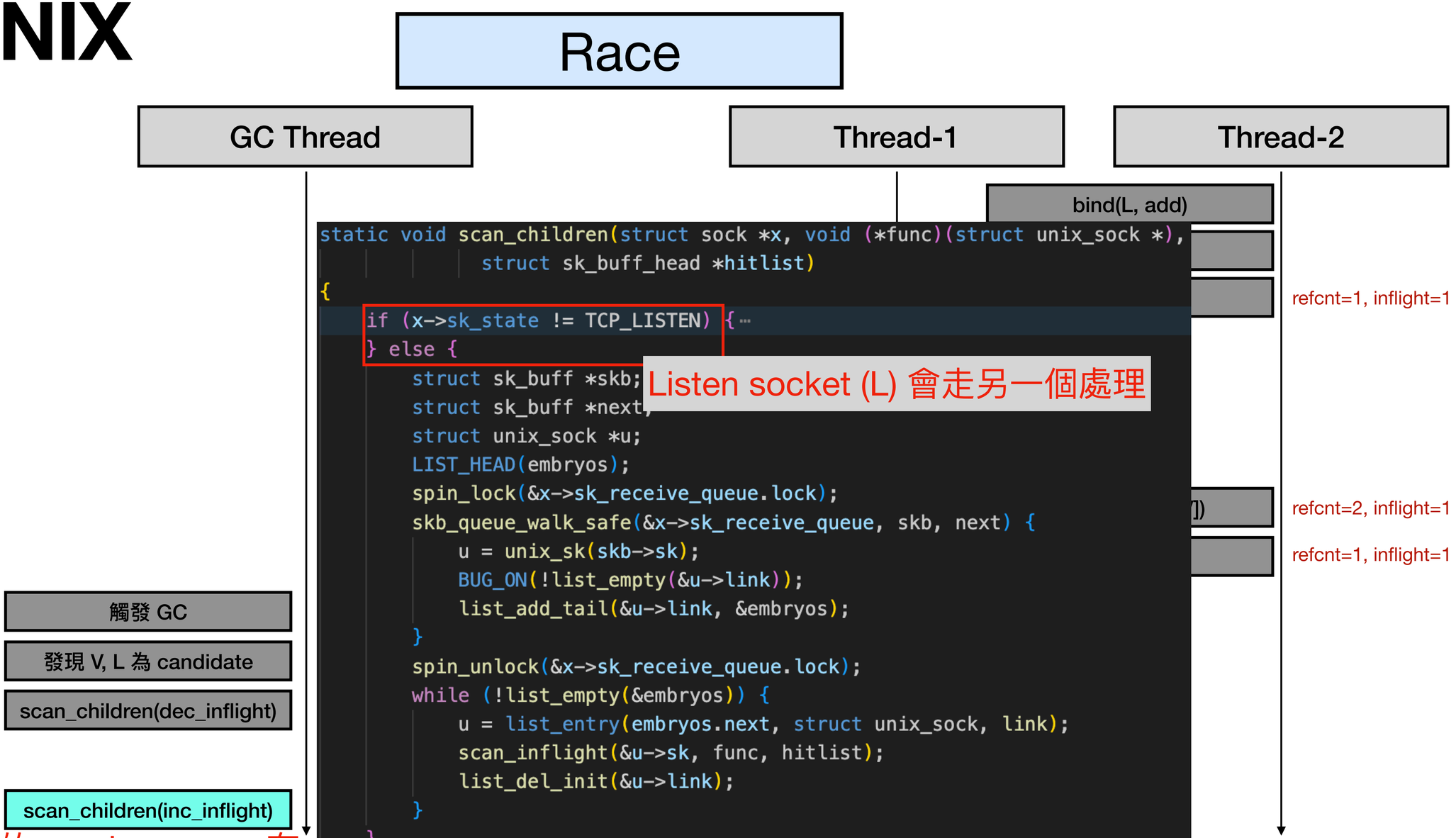
240409



7. L 的 receive queue 有 skb，因此會開始處理

AF_UNIX

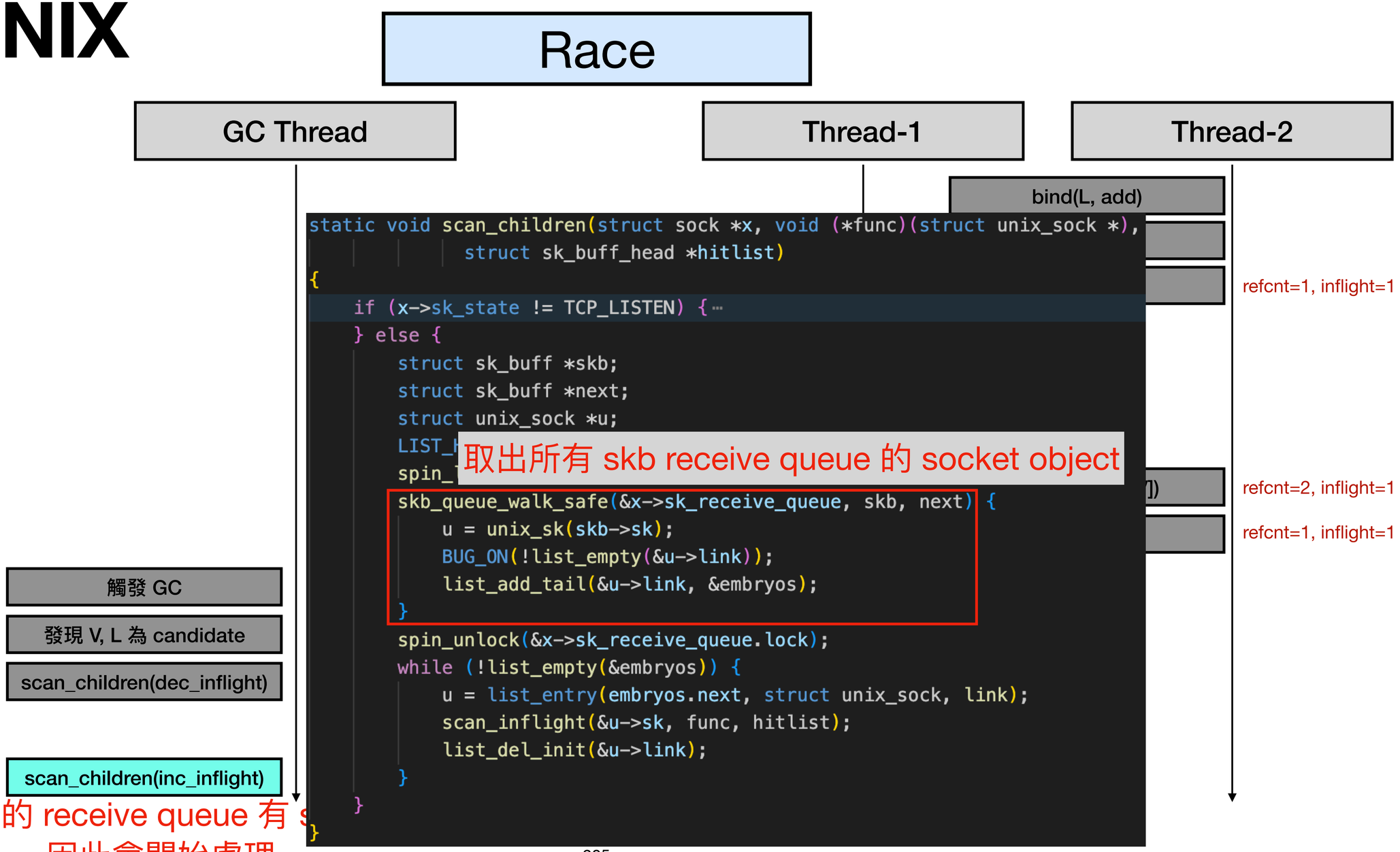
240409




```
static void scan_children(struct sock *x, void (*func)(struct unix_sock *),
                          struct sk_buff_head *hitlist)
{
    if (x->sk_state != TCP_LISTEN) {…
    } else {
        struct sk_buff *skb;
        struct sk_buff *next;
        struct unix_sock *u;
        LIST_HEAD(embryos);
        spin_lock(&x->sk_receive_queue.lock);
        skb_queue_walk_safe(&x->sk_receive_queue, skb, next) {
            u = unix_sk(skb->sk);
            BUG_ON(!list_empty(&u->link));
            list_add_tail(&u->link, &embryos);
        }
        spin_unlock(&x->sk_receive_queue.lock);
        while (!list_empty(&embryos)) {
            u = list_entry(embryos.next, struct unix_sock, link);
            scan_inflight(&u->sk, func, hitlist);
            list_del_init(&u->link);
        }
    }
}
```

AF_UNIX

240409



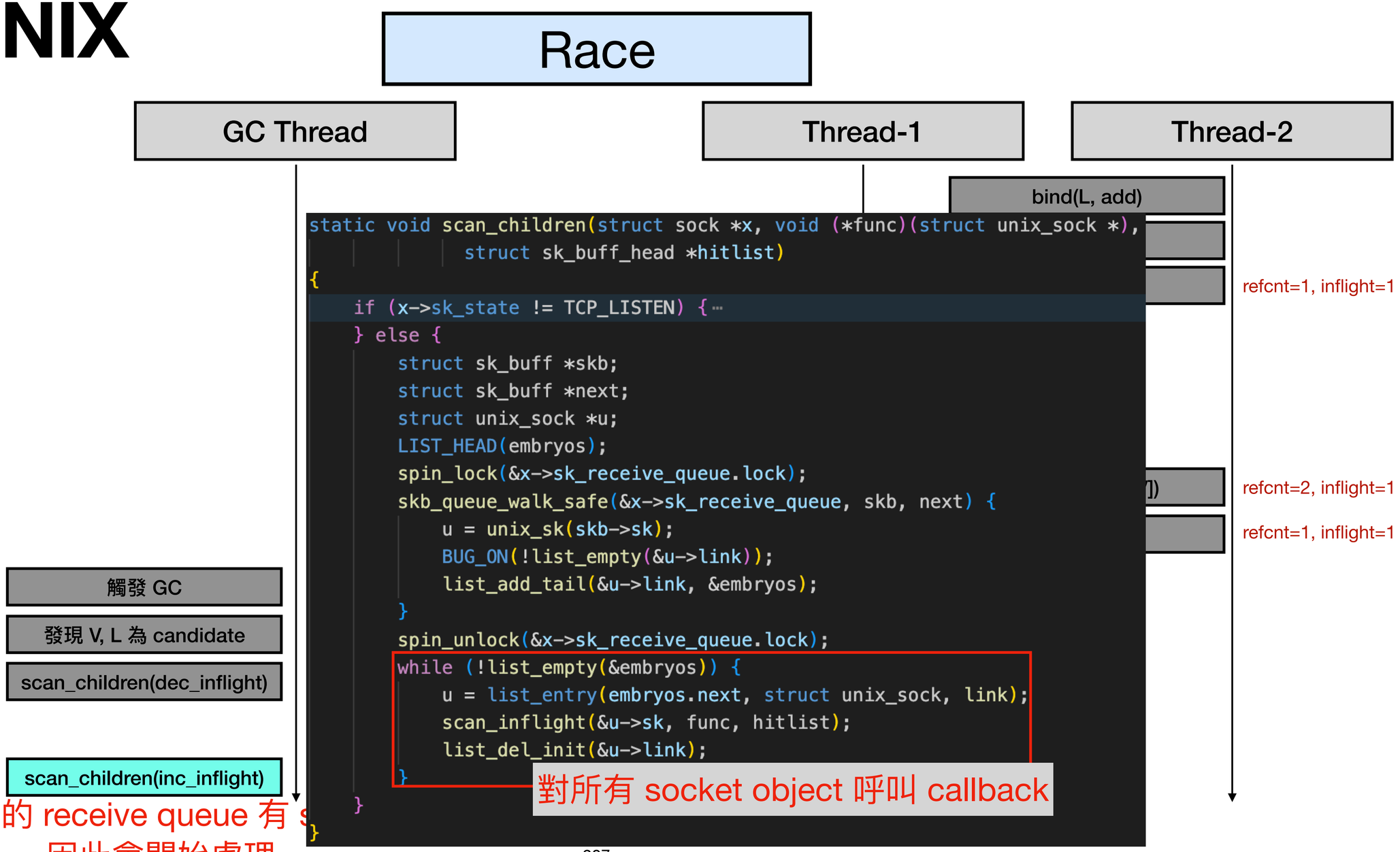
AF_UNIX

240409


```
static void scan_children(struct sock *x, void (*func)(struct unix_sock *),
                          struct sk_buff_head *hitlist)
{
    if (x->sk_state != TCP_LISTEN) {…
    } else {
        struct sk_buff *skb;
        struct sk_buff *next;
        struct unix_sock *u;
        LIST_HEAD(embryos);
        spin_lock(&x->sk_recei
        spin_unlock(&x->sk_receive_queue.lock);
        while (!list_empty(&embryos)) {
            u = list_entry(embryos.next, struct unix_sock, link);
            scan_inflight(&u->sk, func, hitlist);
            list_del_init(&u->link);
        }
    }
}
```

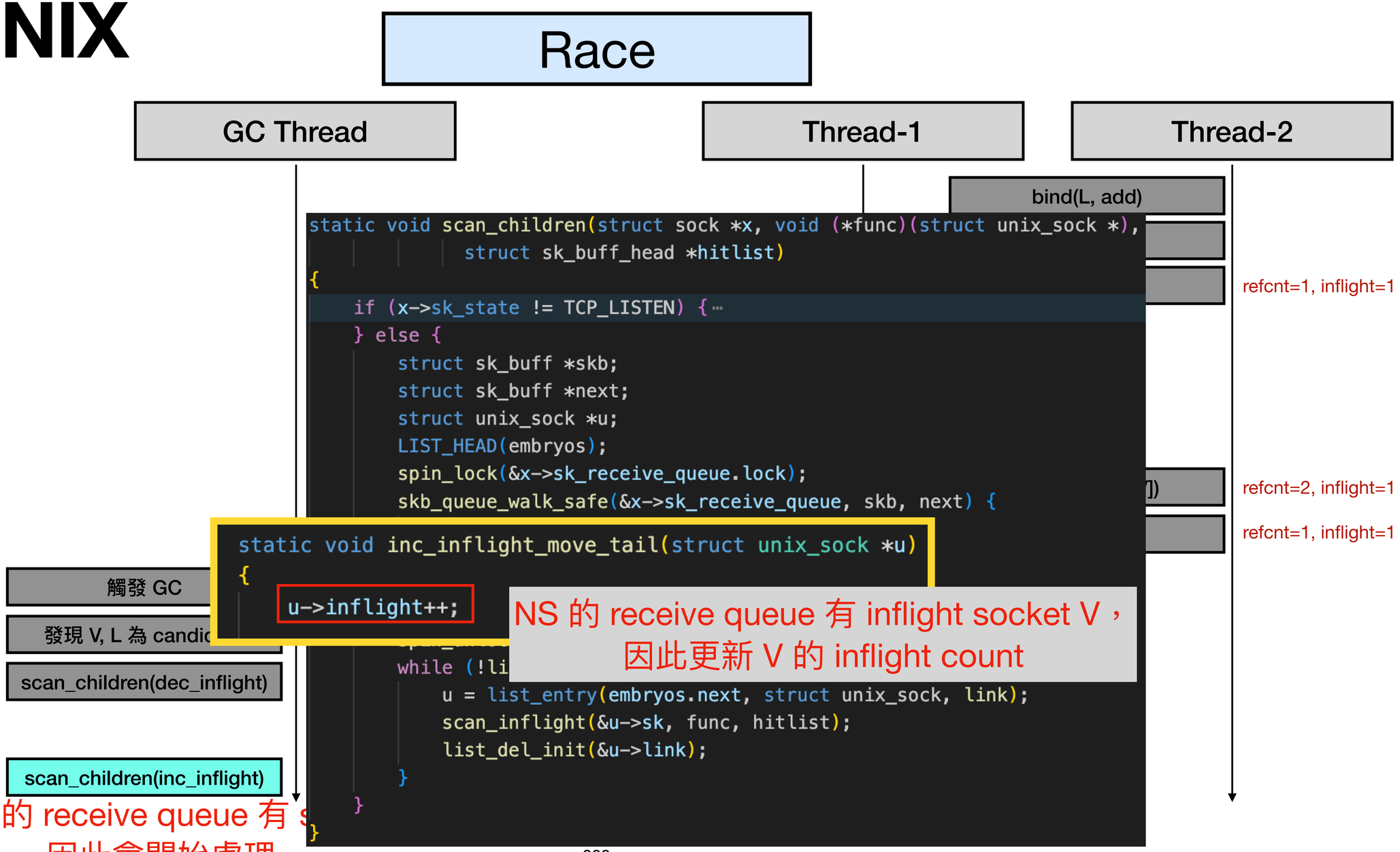
AF_UNIX

240409



AF_UNIX

240409



AF_UNIX

240409

AF_UNIX
240409

- af_unix: Fix garbage collector racing against connect() (CVE-2024-26923)
  - refcnt=1 但 inflight=2 會有什麼影響？
    - [1] 釋放 skb 時也會更新 inflight count，當等於 0 時會 unlink from gc_inflight_list
    - [2] 因為 refcnt = 1，file, socket, sock object 都會被釋放掉，但還在 gc_inflight_list 的 linked list 上

```
void unix_notinflight(struct user_struct *user, struct file *fp)
{
    struct sock *s = unix_get_socket(fp);
    spin_lock(&unix_gc_lock);
    if (s) {
        struct unix_sock *u = unix_sk(s);
        u->inflight--;
        if (!u->inflight)
            list_del_init(&u->link);
        WRITE_ONCE(unix_tot_inflight, unix_tot_inflight - 1);
    }
    WRITE_ONCE(user->unix_inflight, user->unix_inflight - 1);
    spin_unlock(&unix_gc_lock);
}
```

[1]

AF_UNIX
240409

- af_unix: Fix garbage collector racing against connect() (CVE-2024-26923)
  - Patch
    - [1] GC 在檢查 inflight socket 時，如果發現 socket 的 state 為 TCP_LISTEN，就上 lock
    - [2] 連線請求會在 state lock 下 enqueue，因此不會造成 race condition

```c
list_for_each_entry_safe(u, next, &gc_inflight_list, link) {
    struct sock *sk = &u->sk;
    long total_refs;
    total_refs = file_count(sk->sk_socket->file);
    if (total_refs == u->inflight) {
        // [...]
        if (sk->sk_state == TCP_LISTEN) {
            unix_state_lock(sk);
            unix_state_unlock(sk);
        }
    }
}
```

[1]

```c
static int unix_stream_connect(struct socket *sock, struct sockaddr *uaddr,
                               int addr_len, int flags)
{
    /* Latch state of peer */
    unix_state_lock(other);
    // [...]
    spin_lock(&other->sk_receive_queue.lock);
    __skb_queue_tail(&other->sk_receive_queue, skb);
    spin_unlock(&other->sk_receive_queue.lock);
    unix_state_unlock(other);
```

[2]

AF_UNIX

240516

- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - OOB - Out-Of-Band data
    - 提供 process 一個緊急通知的機制
    - sendmsg 使用 MSG_OOB flag 發送大小為 1 byte 的任意資料
      - sys_sendmsg handler 會發送 SIGURG 給 receiver process

```
// [...]
sk_send_sigurg(other);
// [...]
```

AF_UNIX

240516

- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - sendmsg
    - [1] OOB packet 的 skb 會先被存到 peer socket 的 oob_skb 當中
      - 如果之前已經有 oob_skb，就會 drop 該 skb 的 refcnt
    - [2] 再被 enqueue 到 peer socket 的 receive queue

```
if (ousk->oob_skb)
    consume_skb(ousk->oob_skb);
WRITE_ONCE(ousk->oob_skb, skb);
```

[1]

```
skb_queue_tail(&other->sk_receive_queue, skb);
```

[2]

AF_UNIX

240516

- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - recvmsg
    - [1] 若有 MSG_OOB，嘗試從 oob_skb 拿 skb，更新 refcnt 並讀資料
    - [2] 如果從 receive queue 拿到的 skb = oob_skb，unlink from receive queue 並把 oob_skb 設成 NULL

```
oob_skb = u->oob_skb;
// [...]
skb_get(oob_skb);
chunk = state->recv_actor(oob_skb, 0, chunk, state);
consume_skb(oob_skb);
```

[1]

```
if (skb == u->oob_skb) {
    // [...]
    else {
        skb_unlink(skb, &sk->sk_receive_queue);
        WRITE_ONCE(u->oob_skb, NULL);
        if (!WARN_ON_ONCE(skb_unref(skb)))
            kfree_skb(skb);
        skb = skb_peek(&sk->sk_receive_queue);
```

[2]

AF_UNIX

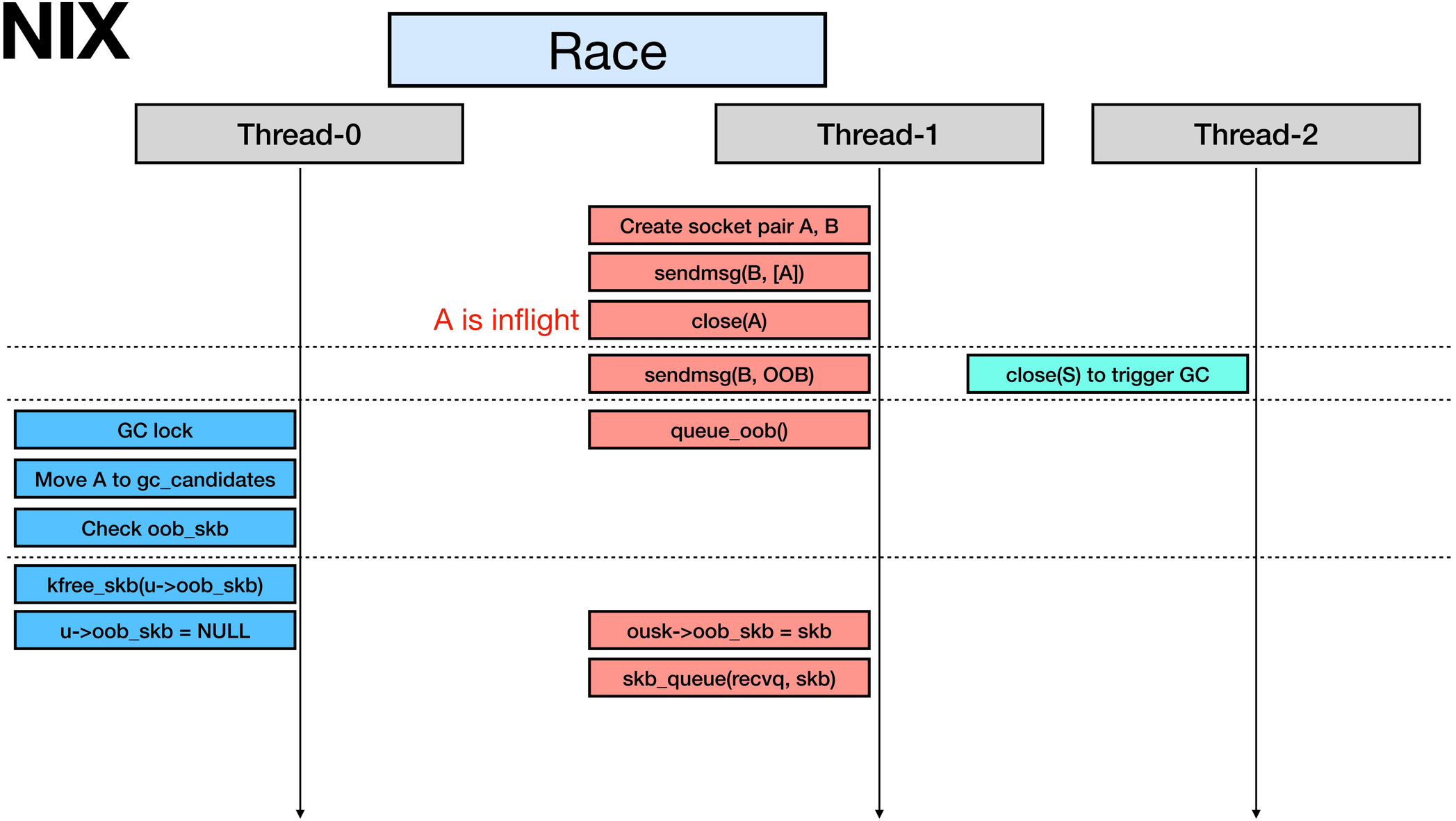
240516

- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - GC 在取出 gc_candidates socket 的 skb 時，如果發現該 socket 有尚未處理的 oob_skb，就會在沒上 lock 的情況下 drop refcnt 並更新成 NULL

```
    skb_queue_head_init(&hitlist);
    list_for_each_entry(u, &gc_candidates, link) {
        scan_children(&u->sk, inc_inflight, &hitlist);

#if !IS_ENABLED(CONFIG_AF_UNIX_OOB)
        if (u->oob_skb) {
            kfree_skb(u->oob_skb);
            u->oob_skb = NULL;
        }
#endif
    }
```

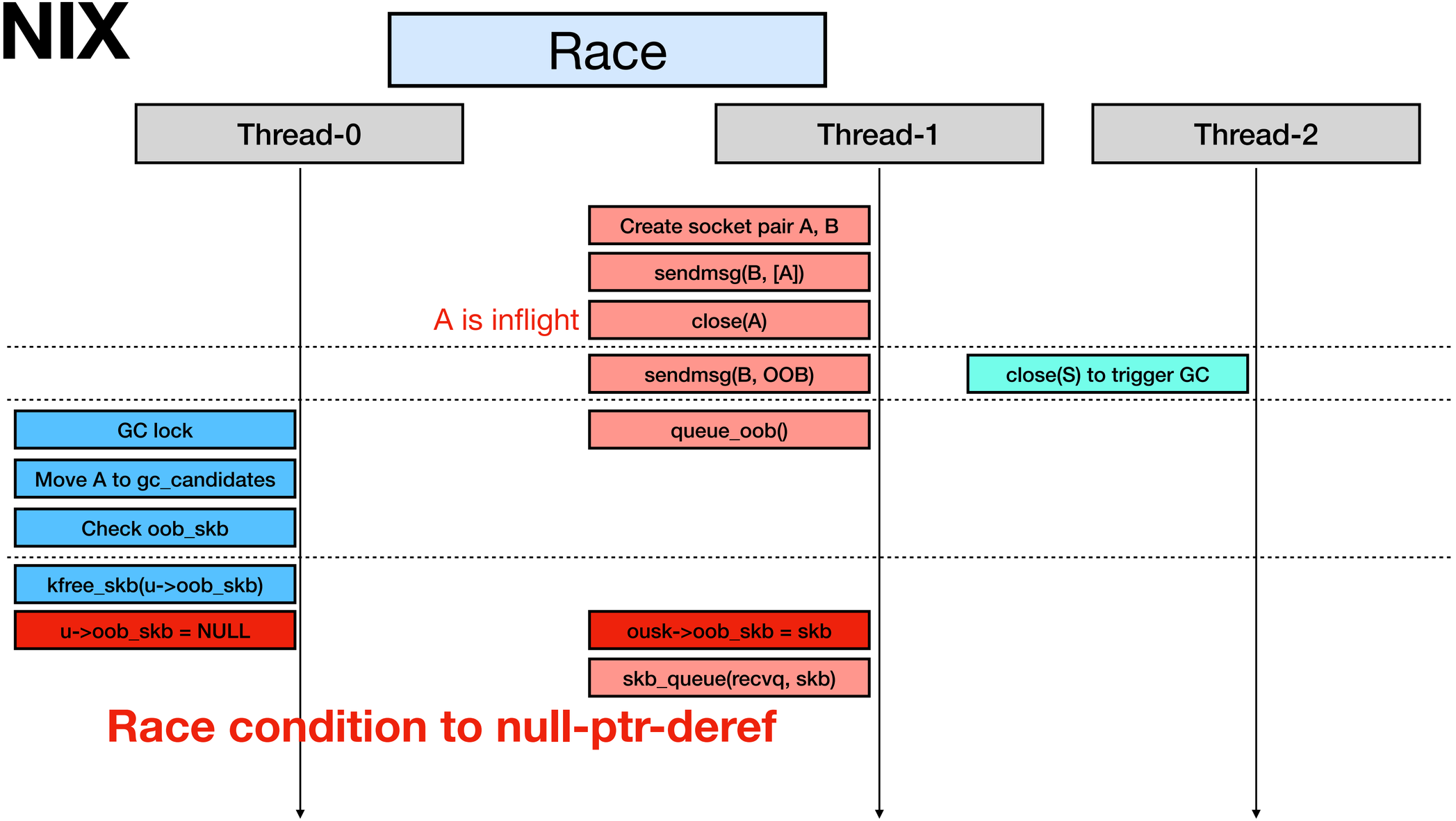
AF_UNIX

240516



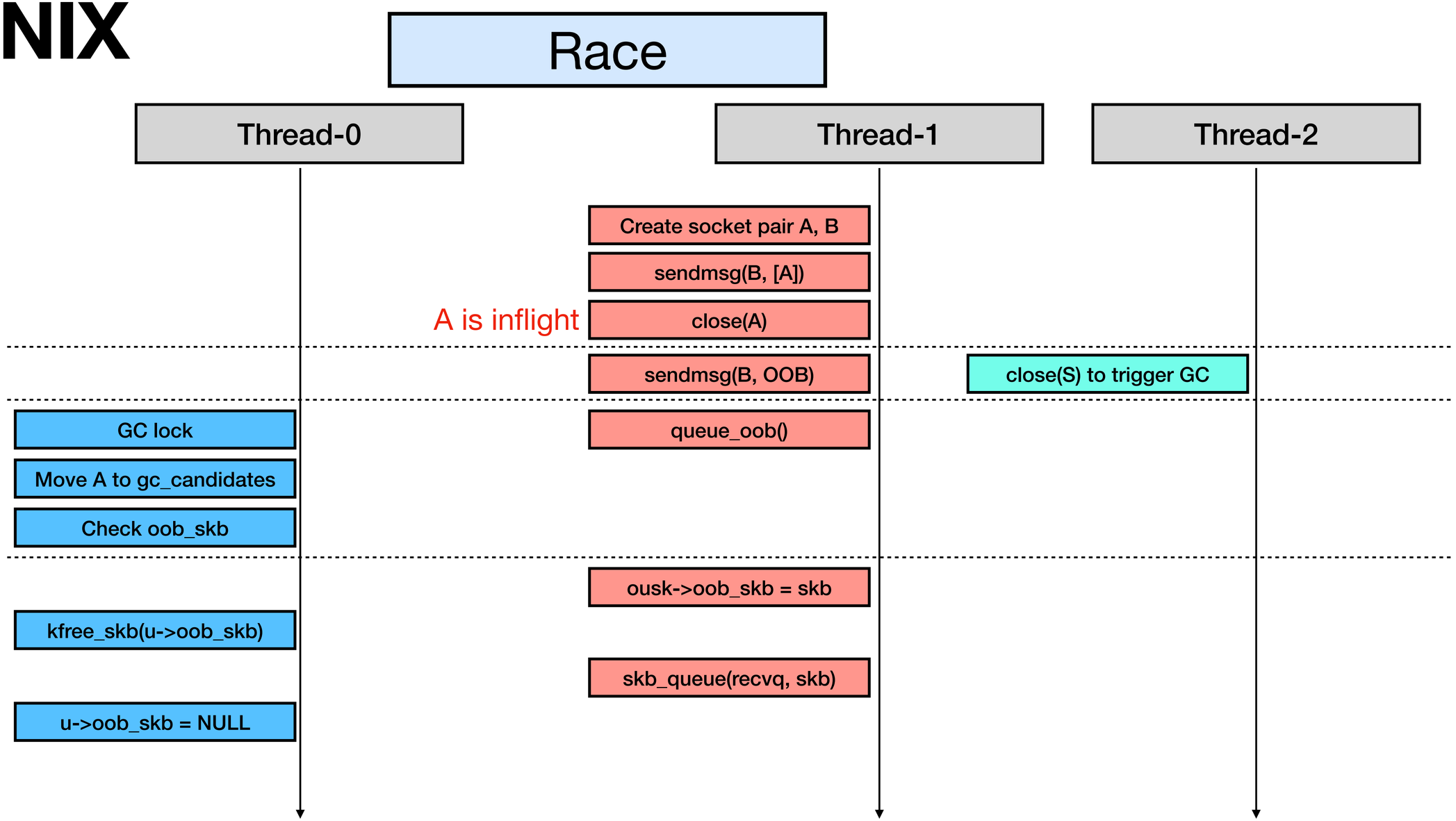
AF_UNIX

240516



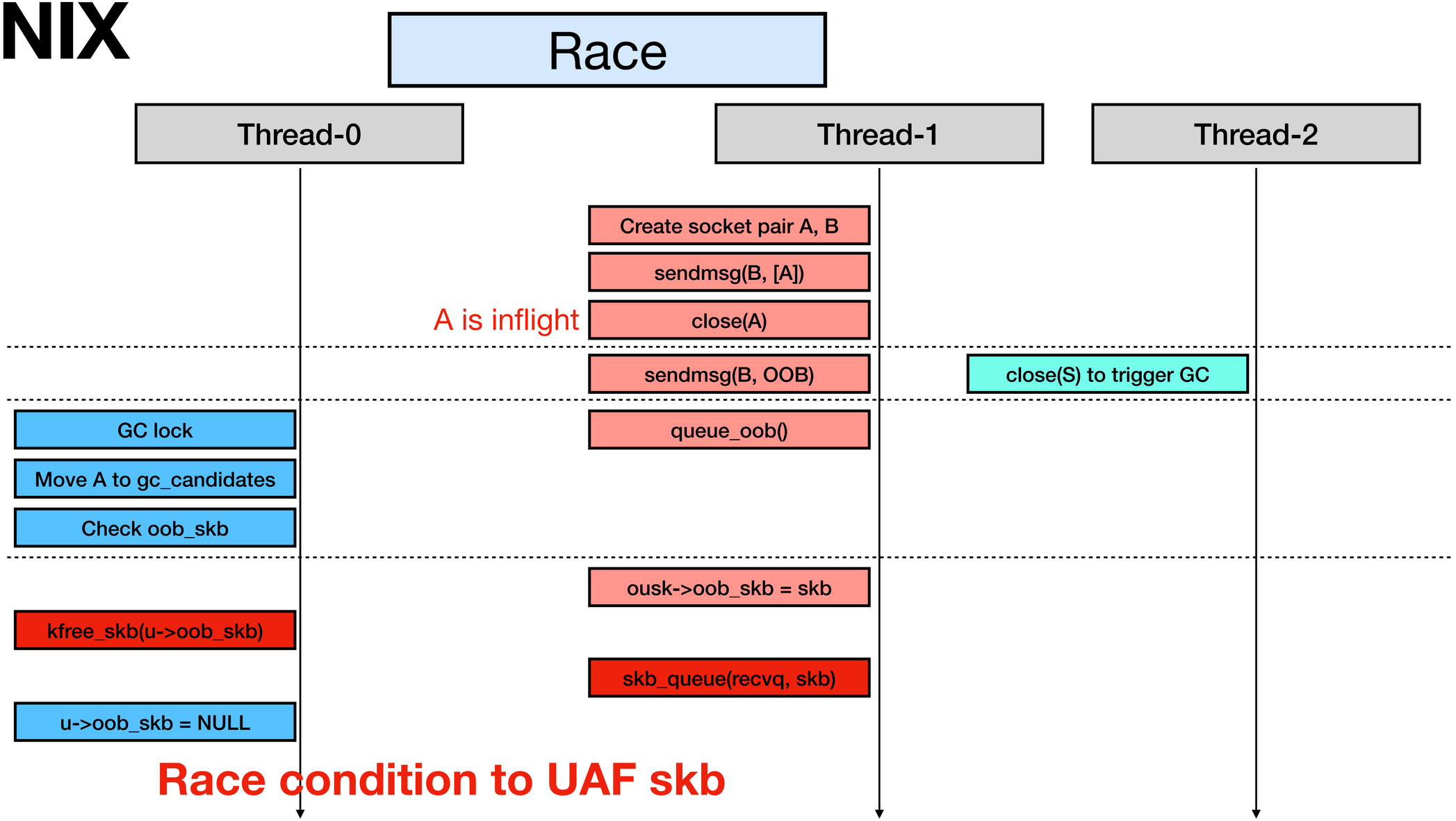
AF_UNIX

240516



AF_UNIX

240516



AF_UNIX

240516

- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - Patch：在所有更新 oob_skb 的地方都加上 receive_queue lock

```
@@ -2171,13 +2171,15 @@ static int queue_oob(struct socket
        maybe_add_creds(skb, sock, other);
        skb_get(skb);

+       scm_stat_add(other, skb);
+
+       spin_lock(&other->sk_receive_queue.lock);
        if (ousk->oob_skb)
                consume_skb(ousk->oob_skb);
-
        WRITE_ONCE(ousk->oob_skb, skb);
+       __skb_queue_tail(&other->sk_receive_queue, skb);
+       spin_unlock(&other->sk_receive_queue.lock);
```

[1] sendmsg with MSG_OOB

```
@@ -2568,8 +2570,10 @@ static int unix_stream_recv_urg(

        mutex_lock(&u->iolock);
        unix_state_lock(sk);
+       spin_lock(&sk->sk_receive_queue.lock);

        if (sock_flag(sk, SOCK_URGINLINE) || !u->oob_sk
+               spin_unlock(&sk->sk_receive_queue.lock)
                unix_state_unlock(sk);
                mutex_unlock(&u->iolock);
                return -EINVAL;
@@ -2581,6 +2585,8 @@ static int unix_stream_recv_urg(s
                WRITE_ONCE(u->oob_skb, NULL);
        else
                skb_get(oob_skb);
+
+       spin_unlock(&sk->sk_receive_queue.lock);
```

[2] recvmsg with MSG_OOB

AF_UNIX

240516

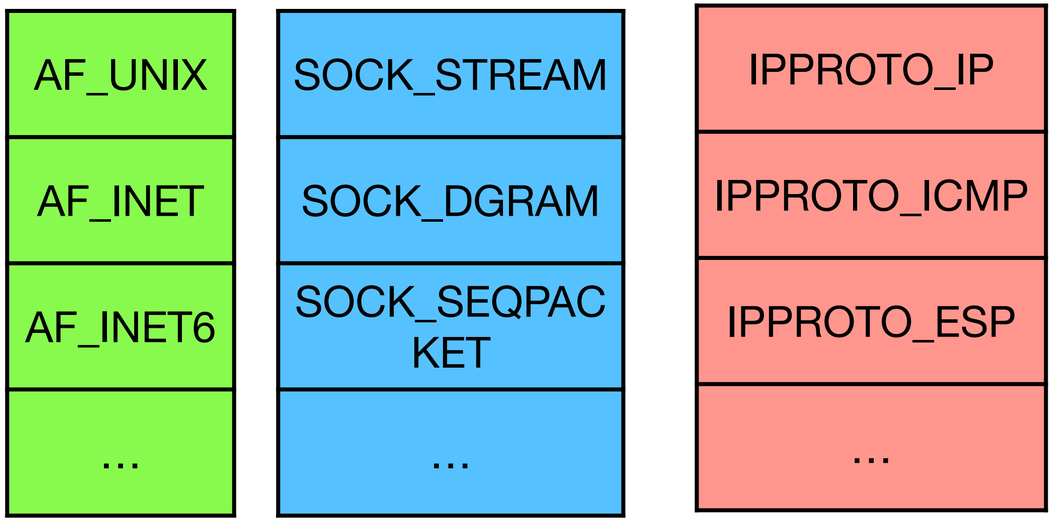
- af_unix: Update unix_sk(sk)->oob_skb under sk_receive_queue lock (CVE-2024-36972)
  - Patch：在所有更新 oob_skb 的地方都加上 receive_queue lock

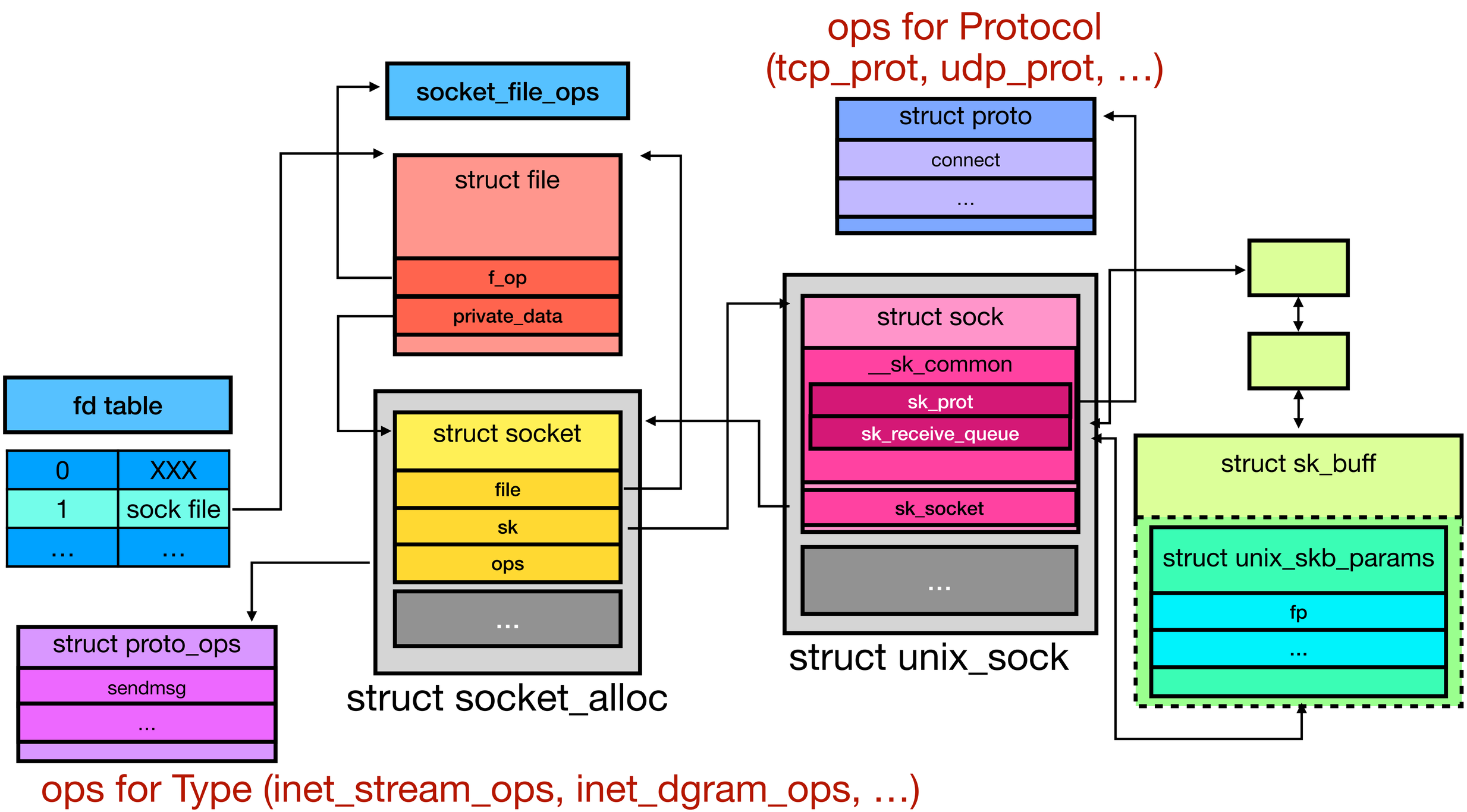
```
@@ -2609,6 +2615,10 @@ static struct sk_buff *manage_oob(struct sk_buff *sk
                consume_skb(skb);
                skb = NULL;
        } else {
+               struct sk_buff *unlinked_skb = NULL;
+
+               spin_lock(&sk->sk_receive_queue.lock);
+
                if (skb == u->oob_skb) {
                        if (copied) {
                                skb = NULL;
@@ -2620,13 +2630,19 @@ static struct sk_buff *manage_oob(struct sk_buff *s
                        } else if (flags & MSG_PEEK) {
                                skb = NULL;
                        } else {
-                               skb_unlink(skb, &sk->sk_receive_queue);
+                               __skb_unlink(skb, &sk->sk_receive_queue);
                                WRITE_ONCE(u->oob_skb, NULL);
-                               if (!WARN_ON_ONCE(skb_unref(skb)))
-                                       kfree_skb(skb);
+                               unlinked_skb = skb;
                                skb = skb_peek(&sk->sk_receive_queue);
                        }
                }
+
+               spin_unlock(&sk->sk_receive_queue.lock);
```

[3] recvmsg

Takeways

socket(family, type, protocol)




ops for Protocol
(tcp_prot, udp_prot, …)
socket_file_ops
struct proto
connect
…
struct file
f_op
private_data
fd table
0
XXX
1
sock file
…
…
struct socket
file
sk
ops
…
struct sock
__sk_common
sk_prot
sk_receive_queue
sk_socket
…
struct sk_buff
struct unix_skb_params
fp
…
struct proto_ops
sendmsg
…
struct socket_alloc
struct unix_sock
ops for Type (inet_stream_ops, inet_dgram_ops, …)