Fall '20 COSE322-00

System Programming

Practice 06. Socket, INET Layer

2020. 11. 12.

Contents

- **Socket Initialization**
  - 소켓과 INET
  - 소켓과 관련된 구조체
  - 소켓 초기화 함수 루틴 분석
  - 소켓과 소켓 버퍼의 관계

- **소켓 버퍼**
  - 소켓 버퍼의 구조 및 필드
  - 소켓 버퍼를 관리 및 조작하는 함수들
  - 소켓 버퍼 헤드의 구조 및 필드
  - 소켓 버퍼 헤드를 관리 및 조작하는 함수들

Socket Initialization

- 소켓과 INET
- 소켓과 관련된 구조체
- 소켓 초기화 함수 루틴 분석
- 소켓과 소켓 버퍼의 관계

Socket과 INET

- **Socket은 End-to-end 연결의 단위**
  - 시스템은 소켓이 생성된 후 해당되는 PF에 따라 적합하게 소켓과 관련된 자료구조를 생성하고 초기화함
- **INET Socket**
  - 소켓이 PF_INET, 즉 TCP/IP 프로토콜 스택을 사용하는 경우임
- **Transport Layer의 TCP에 도달하기 직전까지의 과정**

struct socket

❖ BSD socket (interface)

struct socket

include/linux/net.h

```
110 struct socket {
111     socket_state            state;
112
113     short                   type;
114
115     unsigned long           flags;
116
117     struct socket_wq __rcu  *wq;
118
119     struct file             *file;
120     struct sock             *sk;
121     const struct proto_ops  *ops;
122 };
```

SS_FREE / SS_UNCONNECTED / SS_CONNECTING / SS_CONNECTED / SS_DISCONNECTING

SOCK_STREAM, SOCK_DGRAM, SOCK_RAW

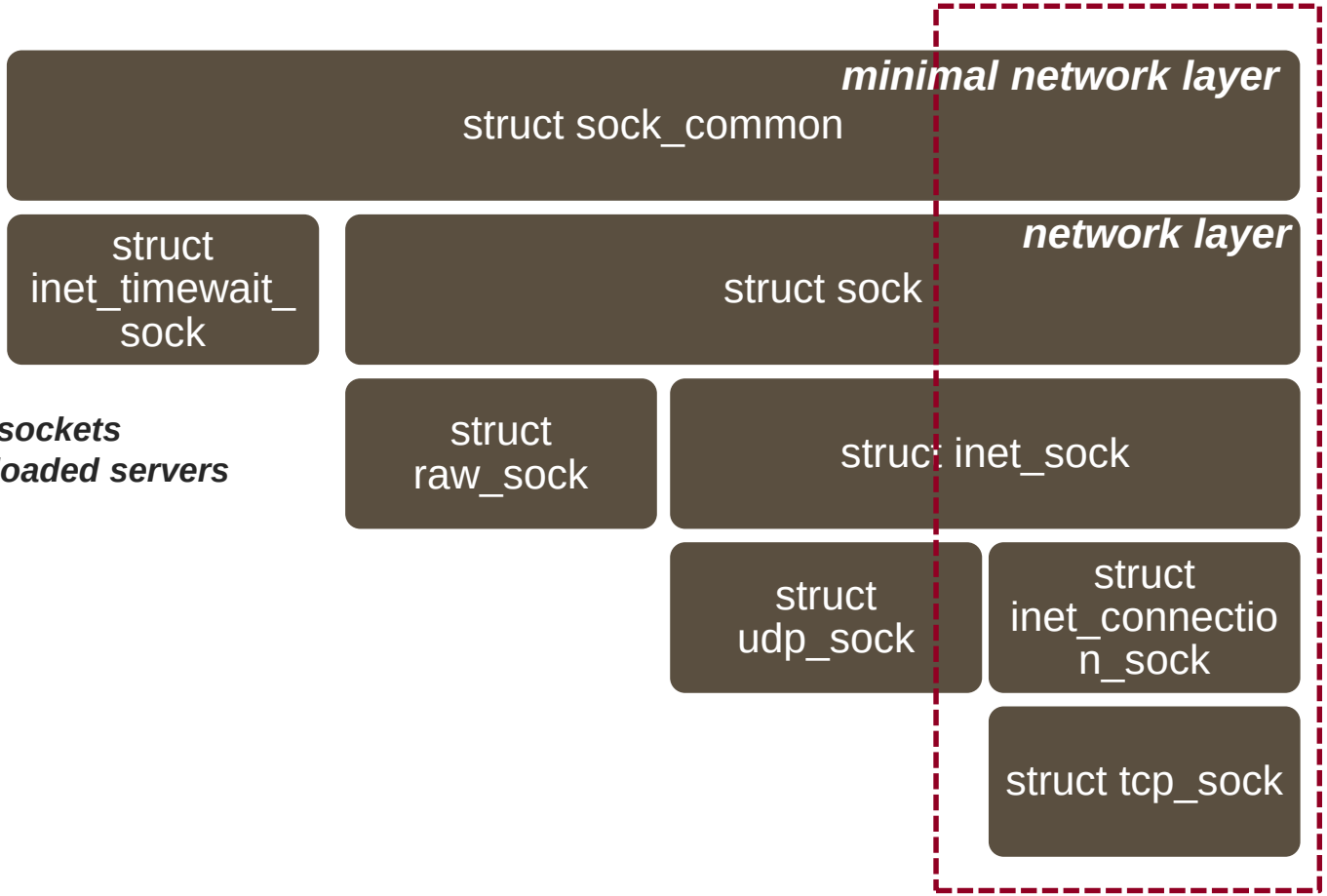
버퍼가 Full인지 표시하는 flag

socke의 event를 기다리는 thread의 wait queue

Protocol family specific ops

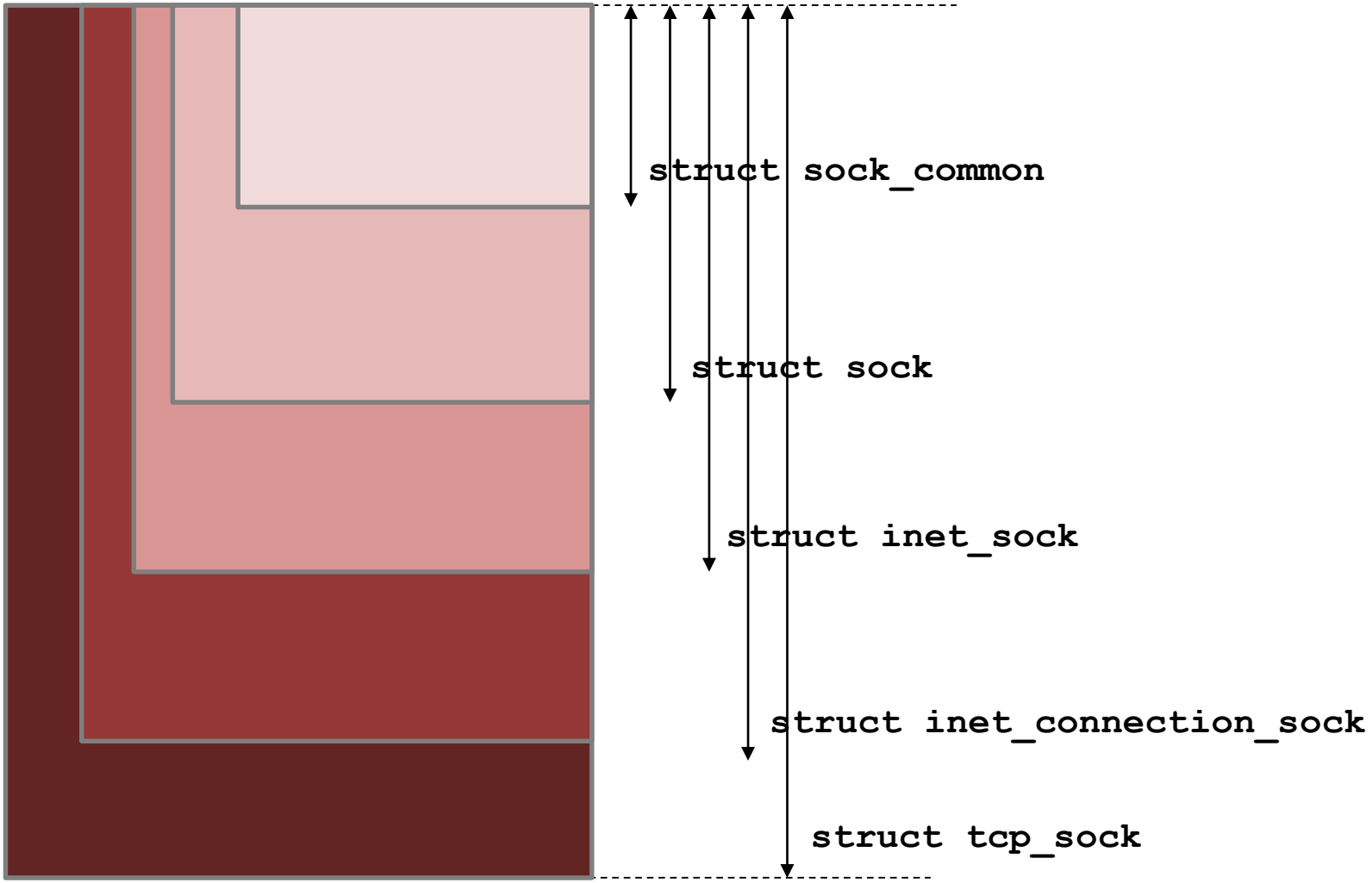
struct sock

- **Polymorphism in Object-oriented!**
  - 만약 TCP 소켓이라면!



works around the memory consumption problems of sockets in such a state on heavily loaded servers

struct sock hierarchy



struct sock

struct sock_common

- 포트번호
- PF
- State
- skc_prot
  - pf 내부 프로토콜 핸들러
- tx 큐의 수
- refcnt

struct sock

- struct sock_common
- sk_buff_head
- snd, rcv 버퍼 크기
- sock wait queue
- 사용하는 프로토콜 (pf 내)
- sk_type (stream or dgram)
- 여러 콜백함수들

struct inet_sock

- struct sock
- 소스 IP주소, 포트
- TTL

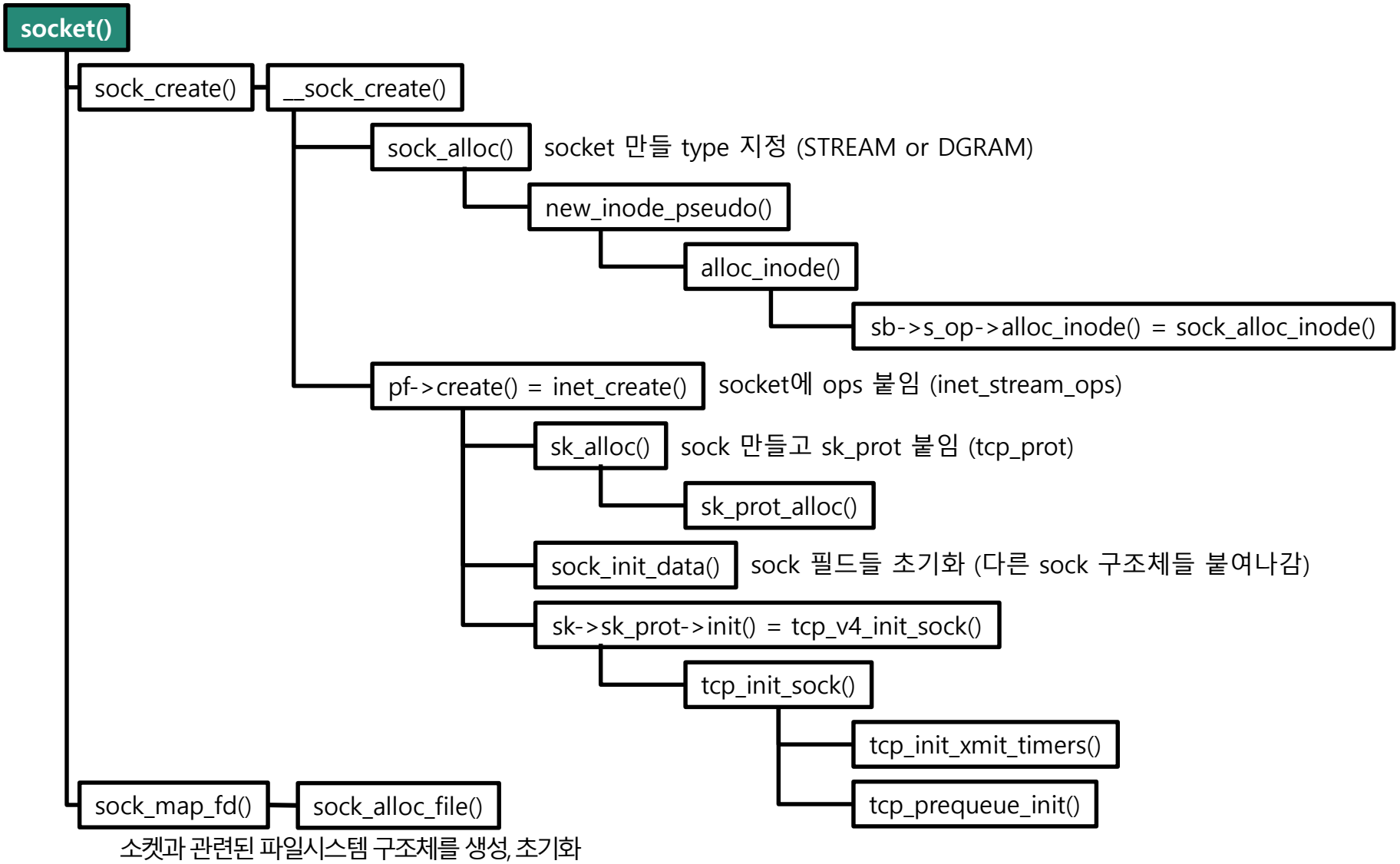
struct inet_connection_sock

- struct inet_sock
- 각종 타이머 (재전송, 지연 측정 등)
- congestion control state
- 통계정보
- congestion control 관련 ops
- stream 프로토콜 관련 ops

struct tcp_sock

- struct inet_connection_sock
- prequeue
- seq / ack 관련 변수
- RTT 측정을 위한 변수
- congestion control, slow start를 위한 변수들

sys_socketcall() call chain



sys_socketcall()

❖ **User API에서 Socket creation을 할 경우 수행됨**

– 우리는 PF_INET, TCP 기준으로 코드를 따라감 (Linux Kernel 4.4기준)

• `socket(PF_INET, SOCK_STREAM, 0)`

sys_socketcall()

net/socket.c

```
1213 SYSCALL_DEFINE3(socket, int, family, int, type, int, protocol)
1214 {
1215     int retval;
1216     struct socket *sock;
1217     int flags;
 …
1233     retval = sock_create(family, type, protocol, &sock);
1234     if (retval < 0)
1235         goto out;
1236
1237     retval = sock_map_fd(sock, flags & (O_CLOEXEC | O_NONBLOCK));
1238     if (retval < 0)
1239         goto out_release;
 …
```

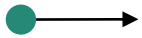
socket과 sock 구조체를 생성하고 초기화하는 함수

소켓과 관련된 파일시스템 구조체를 생성, 초기화

Up to now

sys_socketcall()

struct socket *sock



socket() 시스템콜에서 socket 구조체의 포인터가 선언 됨

sock_create()

sock_create()

net/socket.c

```
1201 int sock_create(int family, int type, int protocol, struct socket **res)
1202 {
1203     return __sock_create(current->nsproxy->net_ns, family, type,
                             protocol, res, 0);
1204 }
```

family: PF_INET
type: SOCK_STREAM
protocol: 0
res: 앞서 선언한 소켓

__sock_create()

__sock_create()

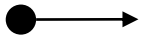
net/socket.c

PF_INET

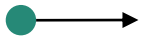
```
1088 int __sock_create(struct net *net, int family, int type, int protocol,
1089                   struct socket **res, int kern)   SOCK_STREAM   0
1090 {                              앞서 선언한 소켓
1091     int err;
1092     struct socket *sock;
1093     const struct net_proto_family *pf;
 …
1127     sock = sock_alloc();
 …
1134     sock->type = type;
 …
1148     pf = rcu_dereference(net_families[family]);
 …
1163     err = pf->create(net, sock, protocol, kern);
 …
1182     *res = sock;
 …
1198 }
```

Up to now

sys_socketcall()
struct socket *sock



__sock_create()
struct socket *sock



socket() 시스템콜에서 socket 구조체의 포인터가 선언 됨

sock_alloc()

sock_alloc()

net/socket.c

```
536 static struct socket *sock_alloc(void)
537 {
538     struct inode *inode;
539     struct socket *sock;
540
541     inode = new_inode_pseudo(sock_mnt->mnt_sb);
 …
545     sock = SOCKET_I(inode);
 …       ~ inode 필드 몇 가지 초기화 ~
555     return sock;
```

VFS에 소켓에 대한 파일시스템 Entry 추가 및 socket의 필드 초기화 수행 (소켓을 마운트하는 슈퍼블록)

new_inode_pseudo()

new_inode_pseudo()

fs/inode.c

```
866 /**
867  *      new_inode_pseudo        - obtain an inode
868  *      @sb: superblock
869  *
870  *      Allocates a new inode for given superblock.
 ...
876 struct inode *new_inode_pseudo(struct super_block *sb)
877 {
878         struct inode *inode = alloc_inode(sb);
879                                         sock_mnt->mnt_sb
880         if (inode) {
881                 spin_lock(&inode->i_lock);
882                 inode->i_state = 0;
883                 spin_unlock(&inode->i_lock);
884                 INIT_LIST_HEAD(&inode->i_sb_list);
885         }
886         return inode;
887 }
```

alloc_inode()

alloc_inode()

fs/inode.c

```
193 static struct inode *alloc_inode(struct super_block *sb)
194 {                                          sock_mnt->mnt_sb
195     struct inode *inode;
196
197     if (sb->s_op->alloc_inode)
198         inode = sb->s_op->alloc_inode(sb);
199     else
200         inode = kmem_cache_alloc(inode_cachep, GFP_KERNEL);
 …
205     if (unlikely(inode_init_always(sb, inode))) {
206         if (inode->i_sb->s_op->destroy_inode)
207             inode->i_sb->s_op->destroy_inode(inode);
208         else
209             kmem_cache_free(inode_cachep, inode);
210         return NULL;
211     }
212
213     return inode;
214 }
```

무슨함수가실행될것인가?

무슨 함수가 실행되나?

net/socket.c

```
330 static struct vfsmount *sock_mnt __read_mostly;
```

```
2482 static int __init sock_init(void)
2483 {
 ...
2506     sock_mnt = kern_mount(&sock_fs_type);
 ...
2530 }
```

kern_mount()에서 vfs_kern_mount()를 호출
→ 마운트될 때 파일시스템(sock_fs_type)의 mount 함수가 호출됨

```
332 static struct file_system_type sock_fs_type = {
333     .name =         "sockfs",
334     .mount =        sockfs_mount,
335     .kill_sb =      kill_anon_super,
336 };
```

```
323 static struct dentry *sockfs_mount(struct file_system_type *fs_type,
324                            int flags, const char *dev_name, void *data)
325 {
326     return mount_pseudo(fs_type, "socket:", &sockfs_ops,
327         &sockfs_dentry_operations, SOCKFS_MAGIC);
328 }
```

```
304 static const struct super_operations sockfs_ops = {
305     .alloc_inode    = sock_alloc_inode,
 ...
308 };
```

sock_alloc_inode()

sock_alloc_inode()

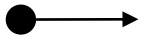
net/socket.c

```
245 static struct inode *sock_alloc_inode(struct super_block *sb)
246 {
247     struct socket_alloc *ei;
248     struct socket_wq *wq;
249
250     ei = kmem_cache_alloc(sock_inode_cachep, GFP_KERNEL);
251     if (!ei)
252         return NULL;
253     wq = kmalloc(sizeof(*wq), GFP_KERNEL);
258     init_waitqueue_head(&wq->wait);
259     wq->fasync_list = NULL;
260     RCU_INIT_POINTER(ei->socket.wq, wq);
261
262     ei->socket.state = SS_UNCONNECTED;
263     ei->socket.flags = 0;
264     ei->socket.ops = NULL;
265     ei->socket.sk = NULL;
266     ei->socket.file = NULL;
267
268     return &ei->vfs_inode;
269 }
```

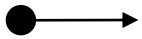
```
1349 struct socket_alloc {
1350     struct socket socket;
1351     struct inode vfs_inode;
1352 };
```

Up to now

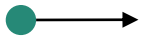
sys_socketcall()
struct socket *sock



__sock_create()
struct socket *sock



sock_alloc()
struct socket *sock



sock_alloc_inode()
struct socket_alloc *ei

state	SS_UNCONNECTED
type	
flags	0
wq	wq
file	NULL
sk	NULL
ops	NULL
inode	

struct socket socket

struct inode vfs_inode

```
1349 struct socket_alloc {
1350     struct socket socket;
1351     struct inode vfs_inode;
1352 };
```

sock_alloc()에서 struct socket *sock 선언

sock_alloc_inode()에서 struct socket_alloc *ei 선언. socket과 inode로 구성됨

Return to alloc_inode()

alloc_inode()

fs/inode.c

```
193 static struct inode *alloc_inode(struct super_block *sb)
194 {                                              sock_mnt->mnt_sb
195     struct inode *inode;
196
197     if (sb->s_op->alloc_inode)
198         inode = sb->s_op->alloc_inode(sb);   ———— 지금까지 이 함수 따라가 봄
199     else
200         inode = kmem_cache_alloc(inode_cachep, GFP_KERNEL);
 …
205     if (unlikely(inode_init_always(sb, inode))) {
206         if (inode->i_sb->s_op->destroy_inode)
207             inode->i_sb->s_op->destroy_inode(inode);
208         else
209             kmem_cache_free(inode_cachep, inode);
210         return NULL;
211     }
212
213     return inode;
214 }
```

inode 필드 초기화 진행,
대부분의 경우 return 0
-> unlikely()

Return to new_inode_pseudo()

new_inode_pseudo()

fs/inode.c

```
866 /**
867  *      new_inode_pseudo        - obtain an inode
868  *      @sb: superblock
869  *
870  *      Allocates a new inode for given superblock.
 …
876 struct inode *new_inode_pseudo(struct super_block *sb)
877 {
878         struct inode *inode = alloc_inode(sb);
879
880         if (inode) {
881                 spin_lock(&inode->i_lock);
882                 inode->i_state = 0;
883                 spin_unlock(&inode->i_lock);
884                 INIT_LIST_HEAD(&inode->i_sb_list);
885         }
886         return inode;
887 }
```

지금까지 이 함수 따라가 봄

Return to sock_alloc()

sock_alloc()

net/socket.c

```
536 static struct socket *sock_alloc(void)
537 {
538     struct inode *inode;
539     struct socket *sock;
540
541     inode = new_inode_pseudo(sock_mnt->mnt_sb);
 …
545     sock = SOCKET_I(inode);
 …       ~ inode 필드 몇 가지 초기화 ~
555     return sock;
```

지금까지 이 함수 따라가 봄

```
1354 static inline struct socket *SOCKET_I (struct inode *inode)
1355 {
1356     return &container_of(inode, struct socket_alloc, vfs_inode)->socket;
1357 }
```

SOCKET_I는 inode의 주소값으로 socket_alloc에서 socket필드 주소를 반환

Up to now

sys_socketcall()
struct socket *sock

__sock_create()
struct socket *sock

sock_alloc()
struct socket *sock

sock_alloc_inode()
struct socket_alloc *ei

state	SS_UNCONNECTED
type	
flags	0
wq	wq
file	NULL
sk	NULL
ops	NULL
inode	

struct socket socket

struct inode vfs_inode

```
1349 struct socket_alloc {
1350     struct socket socket;
1351     struct inode vfs_inode;
1352 };
```

sock_alloc()에서 SOCKET_I(inode) 함수로 socket의 주소값를 가져옴

Return to __sock_create()

__sock_create()

net/socket.c

```
1088 int __sock_create(struct net *net, int family, int type, int protocol,
1089                   struct socket **res, int kern)
1090 {
1091     int err;
1092     struct socket *sock;
1093     const struct net_proto_family *pf;
 …
1127     sock = sock_alloc();
 …
1134     sock->type = type;
 …
1148     pf = rcu_dereference(net_families[family]);
 …
1163     err = pf->create(net, sock, protocol, kern);
 …
1182     *res = sock;
 …
1198 }
```

PF_INET

SOCK_STREAM

0

앞서 선언한 소켓

지금까지 이 함수 따라가 봄

Up to now

sys_socketcall()
struct socket *sock

__sock_create()
struct socket *sock

sock_alloc_inode()
struct socket_alloc *ei

state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	NULL
ops	NULL
inode	

struct socket socket

struct inode vfs_inode

```
1349 struct socket_alloc {
1350     struct socket socket;
1351     struct inode vfs_inode;
1352 };
```

sock_alloc()이 sock의 주소값을 __sock_create()의 *sock에 return

__sock_create()

__sock_create()

net/socket.c

```
1088 int __sock_create(struct net *net, int family, int type, int protocol,
1089                    struct socket **res, int kern)
1090 {
1091     int err;
1092     struct socket *sock;
1093     const struct net_proto_family *pf;
 …
1127     sock = sock_alloc();
 …
1134     sock->type = type;
 …
1148     pf = rcu_dereference(net_families[family]);
 …
1163     err = pf->create(net, sock, protocol, kern);
 …
1182     *res = sock;
 …
1198 }
```

PF_INET

SOCK_STREAM

0

앞서 선언한 소켓

```
921 static struct net_proto_family inet_family_ops = {
922     .family = PF_INET,
923     .create = inet_create,
924     .owner  = THIS_MODULE,
925 };
```

net/ipv4/af_inet.c

inet_create()

inet_create()

net/ipv4/af_inet.c

```
249 static int inet_create(struct net *net, struct socket *sock, int protocol,
250             int kern)
251 {
252     struct sock *sk;
253     struct inet_protosw *answer;
254     struct inet_sock *inet;
255     struct proto *answer_prot;
256     unsigned char answer_flags;
257     int try_loading_module = 0;
258     int err;
 …
266     list_for_each_entry_rcu(answer, &inetsw[sock->type], list) {
 …
 …       /*프로토콜 확인*/
283     }
 …
315     sock->ops = answer->ops;
316     answer_prot = answer->prot;
 …
```

리스트의 entry를 모두 순회하는 함수
answer : loop cursor
두번째 인자 : list의 head
세번째 인자 : list_head의 이름

inet_create()

inetsw_array[]

net/ipv4/af_inet.c

```
994  static struct inet_protosw inetsw_array[] =
995  {
996      {
997          .type =       SOCK_STREAM,
998          .protocol =   IPPROTO_TCP,
999          .prot =       &tcp_prot,
1000         .ops =        &inet_stream_ops,
1001         .flags =      INET_PROTOSW_PERMANENT |
1002                       INET_PROTOSW_ICSK,
1003     },
 ...
 ...
1021     {
1022         .type =       SOCK_RAW,
1023         .protocol =   IPPROTO_IP,    /* wild card */
1024         .prot =       &raw_prot,
1025         .ops =        &inet_sockraw_ops,
1026         .flags =      INET_PROTOSW_REUSE,
1027     }
1028 };
```

부팅시 inetsw_array가 inetsw에 리스트로 등록이 된다

inet_create()

inet_create()

net/ipv4/af_inet.c

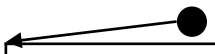
```
249 static int inet_create(struct net *net, struct socket *sock, int
250             int kern)
 …
315     sock->ops = answer->ops; &inet_stream_ops
316     answer_prot = answer->prot; &tcp_prot
 …
323     sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, kern);
 …
331     inet = inet_sk(sk);
 …
```

Up to now

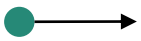
__sock_create()
struct socket *sock

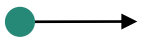
inet_create()
struct sock *sk

inet_create()
struct inet_sock *inet



state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	NULL
ops	&inet_stream_ops





sk_alloc()

sk_alloc()

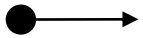
net/core/sock.c

PF_INET

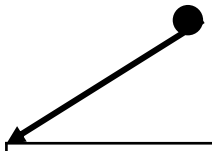
```
1414 struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
1415                       struct proto *prot, int kern)
1416 {                                 &tcp_prot
1417     struct sock *sk;
1418
1419     sk = sk_prot_alloc(prot, priority | __GFP_ZERO, family);
1420     if (sk) {
1421         sk->sk_family = family;
 …
1426         sk->sk_prot = sk->sk_prot_creator = prot;
1427         sock_lock_init(sk);
1428         sk->sk_net_refcnt = kern ? 0 : 1;
1429         if (likely(sk->sk_net_refcnt))
1430             get_net(net);
1431         sock_net_set(sk, net);
1432         atomic_set(&sk->sk_wmem_alloc, 1);
1433
1434         sock_update_classid(sk);
1435         sock_update_netprioidx(sk);
1436     }
1437
1438     return sk;
1439 }
```

Up to now

sys_socketcall()
struct socket *sock

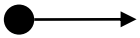


__sock_create()
struct socket *sock

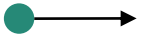


state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	NULL
ops	&inet_stream_ops

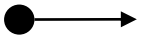
inet_create()
struct sock *sk



sk_alloc()
struct sock *sk



inet_create()
struct inet_sock *inet



sk_alloc()에서 sock 구조체의 포인터를 선언하고 sk_prot_alloc()을 호출함

tcp_prot

tcp_prot

net/ipv4/tcp_ipv4.c

```
2298 struct proto tcp_prot = {
2299     .name           = "TCP",
2300     .owner          = THIS_MODULE,
2301     .close          = tcp_close,
2302     .connect         = tcp_v4_connect,
2303     .disconnect       = tcp_disconnect,
2304     .accept          = inet_csk_accept,
2305     .ioctl          = tcp_ioctl,
2306     .init           = tcp_v4_init_sock,
2307     .destroy         = tcp_v4_destroy_sock,
2308     .shutdown         = tcp_shutdown,
2309     .setsockopt       = tcp_setsockopt,
2310     .getsockopt       = tcp_getsockopt,
2311     .recvmsg         = tcp_recvmsg,
2312     .sendmsg         = tcp_sendmsg,
2313     .sendpage         = tcp_sendpage,
2314     .backlog_rcv       = tcp_v4_do_rcv,
2315     .release_cb       = tcp_release_cb,
2316     .hash           = inet_hash,
2317     .unhash          = inet_unhash,
2318     .get_port         = inet_csk_get_port,
2319     .enter_memory_pressure = tcp_enter_memory_pressure,
2320     .stream_memory_free   = tcp_stream_memory_free,
2321     .sockets_allocated   = &tcp_sockets_allocated,
2322     .orphan_count      = &tcp_orphan_count,
2323     .memory_allocated    = &tcp_memory_allocated,
2324     .memory_pressure    = &tcp_memory_pressure,
2325     .sysctl_mem       = sysctl_tcp_mem,
2326     .sysctl_wmem      = sysctl_tcp_wmem,
2327     .sysctl_rmem      = sysctl_tcp_rmem,
2328     .max_header       = MAX_TCP_HEADER,
2329     .obj_size         = sizeof(struct tcp_sock),
2330     .slab_flags       = SLAB_DESTROY_BY_RCU,
2331     .twsk_prot        = &tcp_timewait_sock_ops,
2332     .rsk_prot         = &tcp_request_sock_ops,
2333     .h.hashinfo       = &tcp_hashinfo,
2334     .no_autobind      = true,
2335 #ifdef CONFIG_COMPAT
2336     .compat_setsockopt   = compat_tcp_setsockopt,
2337     .compat_getsockopt   = compat_tcp_getsockopt,
2338 #endif
2339 #ifdef CONFIG_MEMCG_KMEM
2340     .init_cgroup      = tcp_init_cgroup,
2341     .destroy_cgroup     = tcp_destroy_cgroup,
2342     .proto_cgroup      = tcp_proto_cgroup,
2343 #endif
2344 };
2345 EXPORT_SYMBOL(tcp_prot);
```

return to sk_alloc()

sk_alloc()

net/core/sock.c

```
1414 struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
1415                       struct proto *prot, int kern)
1416 {
1417     struct sock *sk;
1418
1419     sk = sk_prot_alloc(prot, priority | __GFP_ZERO, family);
1420     if (sk) {
1421         sk->sk_family = family;
 …
1426         sk->sk_prot = sk->sk_prot_creator = prot;
1427         sock_lock_init(sk);
1428         sk->sk_net_refcnt = kern ? 0 : 1;
1429         if (likely(sk->sk_net_refcnt))
1430             get_net(net);
1431         sock_net_set(sk, net);
1432         atomic_set(&sk->sk_wmem_alloc, 1);
1433
1434         sock_update_classid(sk);
1435         sock_update_netprioidx(sk);
1436     }
1437
1438     return sk;
1439 }
```

여기까지 따라옴

Up to now

`sys_socketcall()`
struct socket *sock

`inet_create()`
struct sock *sk

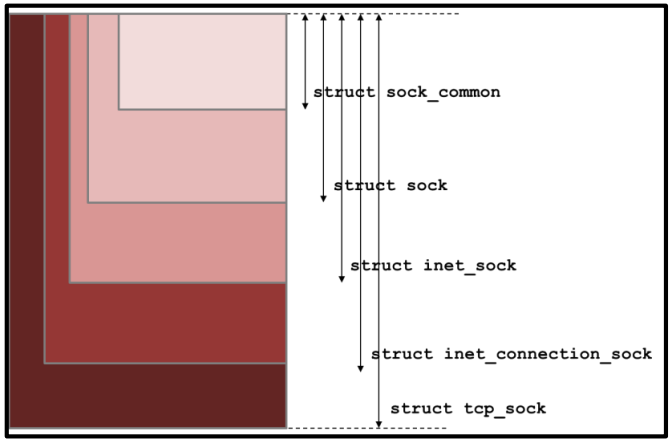
`inet_create()`
struct inet_sock *inet

`__sock_create()`
struct socket *sock

`sk_alloc()`
struct sock *sk

state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	NULL
ops	&inet_stream_ops

struct tcp_sock



`sk_prot_alloc()`함수가 `sk_alloc()`에서 선언한 `struct sock *sk`에 주소 return

return to inet_create()

inet_create()
net/ipv4/af_inet.c

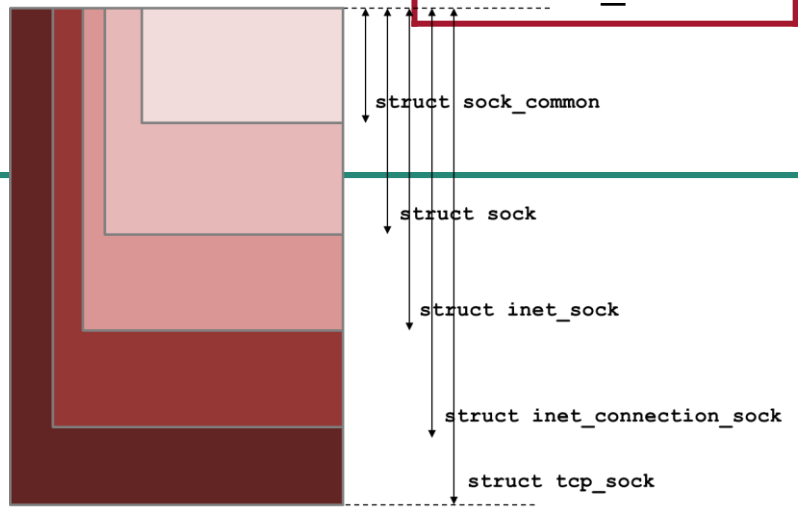
```
249 static int inet_create(struct net *net, struct socket *sock, int proto
250             int kern)
 …
315     sock->ops = answer->ops;
316     answer_prot = answer->prot;
 …
323     sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, kern);
 …
331     inet = inet_sk(sk);
 …
```

여기까지 따라옴

inet_sk()
net/inet_sock.h

```
183 static inline struct inet_sock *inet_sk(const struct sock *sk)
184 {
185     return (struct inet_sock *)sk;
186 }
187
```

Polymorphism in OO concepts!



Up to now

sys_socketcall()
struct socket *sock

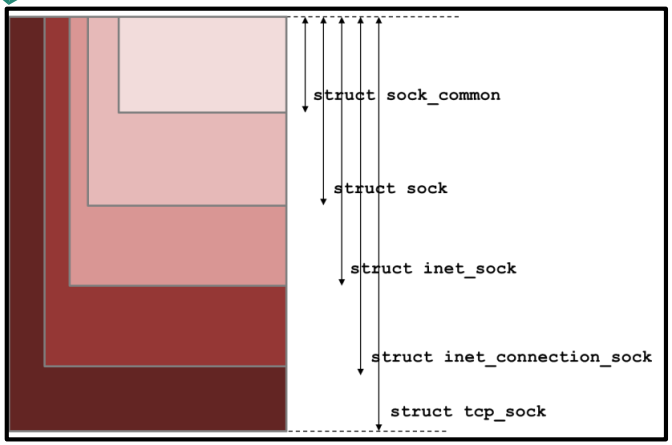
inet_create()
struct sock *sk

inet_create()
struct inet_sock *inet

__sock_create()
struct socket *sock

state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	NULL
ops	&inet_stream_ops

struct tcp_sock



sk_alloc()함수가 inet_create()에서 선언한 struct sock *sk에 주소 return
inet에 sock구조체를 inet_sock 구조체로 type casting하여 주소 복사

inet_create()

inet_create()

net/ipv4/af_inet.c

```
249 static int inet_create(struct net *net, struct socket *sock, int proto
250             int kern)
 …
315     sock->ops = answer->ops;
316     answer_prot = answer->prot;
 …
323     sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, kern);
 …
331     inet = inet_sk(sk);
 …
349     sock_init_data(sock, sk);
350
351     sk->sk_destruct    = inet_sock_destruct;
352     sk->sk_protocol    = protocol;
353     sk->sk_backlog_rcv = sk->sk_prot->backlog_rcv;
354
355     inet->uc_ttl    = -1;
356     inet->mc_loop   = 1;
357     inet->mc_ttl    = 1;
358     inet->mc_all    = 1;
359     inet->mc_index  = 0;
360     inet->mc_list   = NULL;
361     inet->rcv_tos   = 0;
 …
```

sock_init_data()

sock_init_data()
net/core/sock.c

```
2364 void sock_init_data(struct socket *sock, struct sock *sk)
2365 {
2366     skb_queue_head_init(&sk->sk_receive_queue);
2367     skb_queue_head_init(&sk->sk_write_queue);
2368     skb_queue_head_init(&sk->sk_error_queue);
2369
2370     sk->sk_send_head        =       NULL;
2371
2372     init_timer(&sk->sk_timer);
2373
2374     sk->sk_allocation       =       GFP_KERNEL;
2375     sk->sk_rcvbuf           =       sysctl_rmem_default;
2376     sk->sk_sndbuf           =       sysctl_wmem_default;
2377     sk->sk_state            =       TCP_CLOSE;
2378     sk_set_socket(sk, sock);
2379
2380     sock_set_flag(sk, SOCK_ZAPPED);
 ...
```

```
1651 static inline void sk_set_socket(struct sock *sk, struct socket *sock)
1652 {
1653     sk_tx_queue_clear(sk);
1654     sk->sk_socket = sock;
1655 }
```

sk_set_socket()
include/net/sock.h

sock_init_data()

sock_init_data()

net/core/sock.c

```
2364 void sock_init_data(struct socket *sock, struct sock *sk)
2365 {
 …
2382     if (sock) {
2383         sk->sk_type         =       sock->type;
2384         sk->sk_wq           =       sock->wq;
2385         sock->sk            =       sk;
2386     } else
2387         sk->sk_wq           =       NULL;
 …
2394     sk->sk_state_change     =       sock_def_wakeup;
2395     sk->sk_data_ready       =       sock_def_readable;
2396     sk->sk_write_space      =       sock_def_write_space;
2397     sk->sk_error_report     =       sock_def_error_report;
2398     sk->sk_destruct         =       sock_def_destruct;
2399
 …
 …
2428 }
2429 EXPORT_SYMBOL(sock_init_data);
```

callbacks

Rest are initialized at later-stage (connection-setup)

return to inet_create()

inet_create()
net/ipv4/af_inet.c

```
249 static int inet_create(struct net *net, struct socket *sock, int pro
250             int kern)
 …
323     sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, kern);
 …
331     inet = inet_sk(sk);
 …
349     sock_init_data(sock, sk);
350
351     sk->sk_destruct    = inet_sock_destruct;
352     sk->sk_protocol    = protocol;
353     sk->sk_backlog_rcv = sk->sk_prot->backlog_rcv;
354
355     inet->uc_ttl    = -1;
356     inet->mc_loop   = 1;
357     inet->mc_ttl    = 1;
358     inet->mc_all    = 1;
359     inet->mc_index  = 0;
360     inet->mc_list   = NULL;
361     inet->rcv_tos   = 0;
 …
376     if (sk->sk_prot->init) {
377         err = sk->sk_prot->init(sk);
 …
386 }
```

여기까지 따라옴

&tcp_prot

tcp_prot
net/ipv4/tcp_ipv4.c

```
2298 struct proto tcp_prot = {
2299     .name                = "TCP",
2300     .owner               = THIS_MODULE,
2301     .close               = tcp_close,
2302     .connect             = tcp_v4_connect,
2303     .disconnect          = tcp_disconnect,
2304     .accept              = inet_csk_accept,
2305     .ioctl               = tcp_ioctl,
2306     .init                = tcp_v4_init_sock,
2307     .destroy             = tcp_v4_destroy_sock,
 …
```

Up to now

`sys_socketcall()`
struct socket *sock

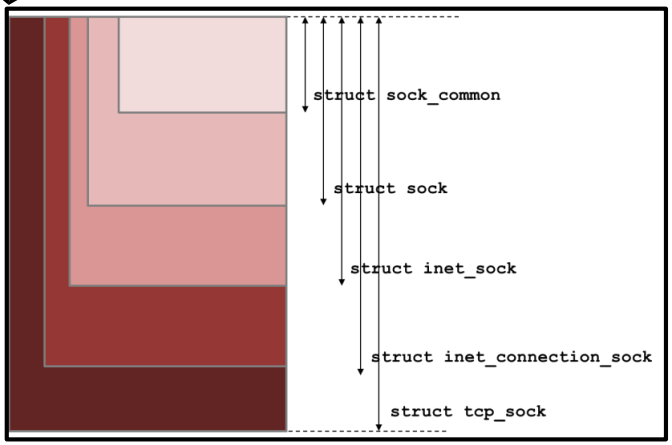
`inet_create()`
struct sock *sk

`inet_create()`
struct inet_sock *inet

`__sock_create()`
struct socket *sock

state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	sk
ops	&inet_stream_ops

struct tcp_sock



`sock_init_data()`에서 `sock->sk`에 sk의 주소 복사

tcp_v4_init_sock()

❖ tcp_prot의 init함수

tcp_prot
net/ipv4/tcp_ipv4.c

```
2298 struct proto tcp_prot = {
2299     .name                   = "TCP",
 …
2306     .init                   = tcp_v4_init_sock,
2307     .destroy                = tcp_v4_destroy_sock,
 …
```

tcp_v4_init_sock()
net/ipv4/tcp_ipv4.c

```
1763 static int tcp_v4_init_sock(struct sock *sk)
1764 {
1765     struct inet_connection_sock *icsk = inet_csk(sk);
1766
1767     tcp_init_sock(sk);
1768
1769     icsk->icsk_af_ops = &ipv4_specific;
 …
1775     return 0;
1776 }
```

inet_csk()
include/net/inet_connection_sock.h

```
147 static inline struct inet_connection_sock *inet_csk(const struct sock *sk)
148 {
149     return (struct inet_connection_sock *)sk;
150 }
```

tcp_init_sock()

tcp_init_sock()
net/ipv4/tcp.c

```
379 void tcp_init_sock(struct sock *sk)
380 {
381     struct inet_connection_sock *icsk = inet_csk(sk);
382     struct tcp_sock *tp = tcp_sk(sk);
383
384     __skb_queue_head_init(&tp->out_of_order_queue);
385     tcp_init_xmit_timers(sk);
386     tcp_prequeue_init(tp);
387     INIT_LIST_HEAD(&tp->tsq_node);
388
389     icsk->icsk_rto = TCP_TIMEOUT_INIT;
390     tp->mdev_us = jiffies_to_usecs(TCP_TIMEOUT_INIT);
 ...
 ...
```

tcp_sk()
include/linux/tcp.h

```
359 static inline struct tcp_sock *tcp_sk(const struct sock *sk)
360 {
361         return (struct tcp_sock *)sk;
362 }
```

tcp_init_xmit_timers()

tcp_init_xmit_timers()

net/ipv4/tcp_timer.c

```
 33 static void tcp_write_timer(unsigned long);
 34 static void tcp_delack_timer(unsigned long);
 35 static void tcp_keepalive_timer (unsigned long data);
 36
 …
659 void tcp_init_xmit_timers(struct sock *sk)
660 {
661     inet_csk_init_xmit_timers(sk, &tcp_write_timer, &tcp_delack_timer,
662                               &tcp_keepalive_timer);
663 }
```

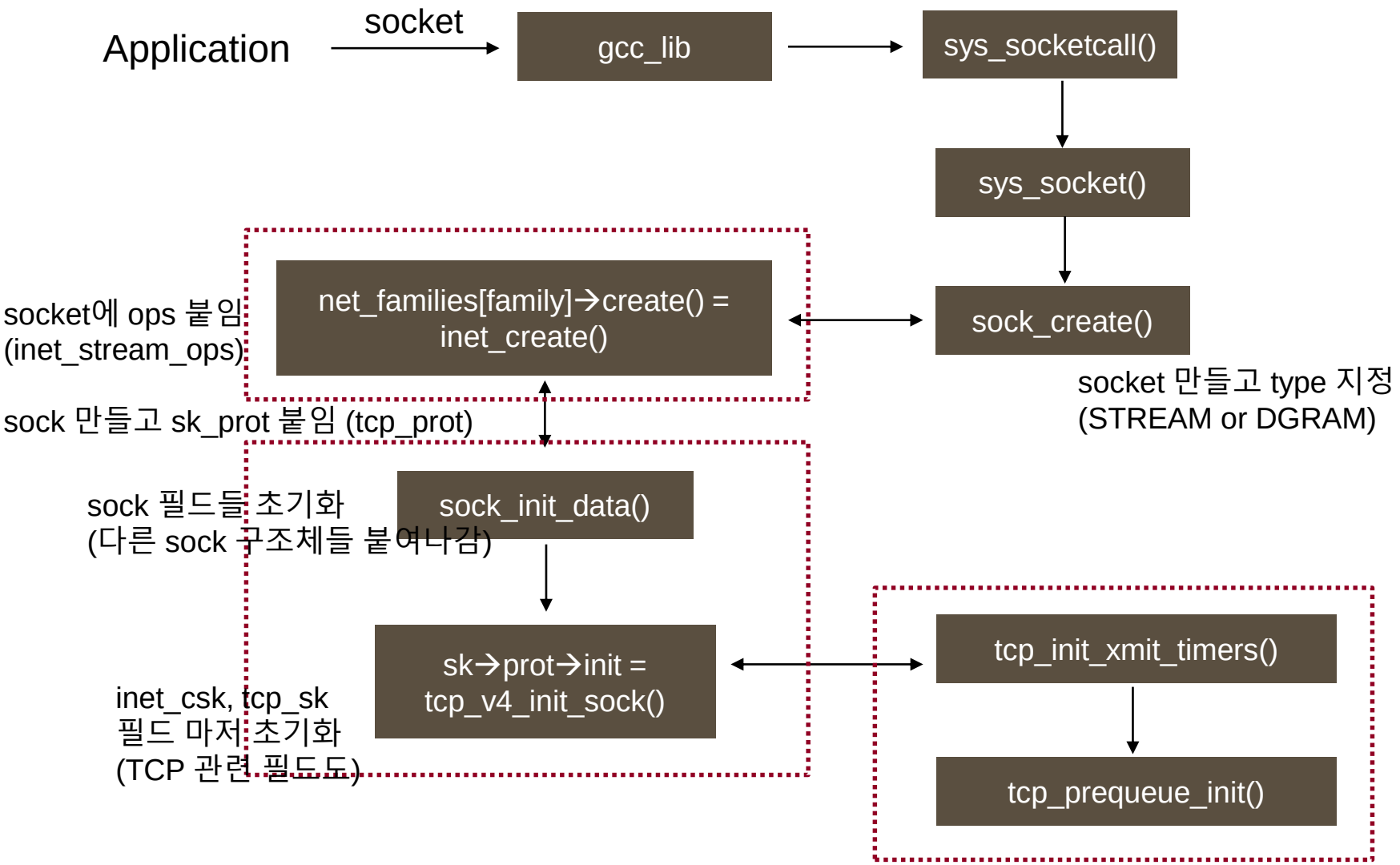
tcp_prequeue_init()

net/tcp.h

```
1148 /* Prequeue for VJ style copy to user, combined with checksumming. */
1149
1150 static inline void tcp_prequeue_init(struct tcp_sock *tp)
1151 {
1152     tp->ucopy.task = NULL;
1153     tp->ucopy.len = 0;
1154     tp->ucopy.memory = 0;
1155     skb_queue_head_init(&tp->ucopy.prequeue);
1156 }
```

ucopy : Direct copy to user를 위한 구조체

Up to here


Application
socket
gcc_lib
sys_socketcall()
sys_socket()
sock_create()
socket 만들고 type 지정
(STREAM or DGRAM)
net_families[family]→create() =
inet_create()
socket에 ops 붙임
(inet_stream_ops)
sock 만들고 sk_prot 붙임 (tcp_prot)
sock_init_data()
sock 필드들 초기화
(다른 sock 구조체들 붙여나감)
sk→prot→init =
tcp_v4_init_sock()
inet_csk, tcp_sk
필드 마저 초기화
(TCP 관련 필드됴)
tcp_init_xmit_timers()
tcp_prequeue_init()

return to inet_create()

inet_create()

net/ipv4/af_inet.c

```
249 static int inet_create(struct net *net, struct socket *sock, int pro
250             int kern)
 …
323     sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, kern);
 …
331     inet = inet_sk(sk);
 …
349     sock_init_data(sock, sk);
350
351     sk->sk_destruct    = inet_sock_destruct;
352     sk->sk_protocol    = protocol;
353     sk->sk_backlog_rcv = sk->sk_prot->backlog_rcv;
354
355     inet->uc_ttl    = -1;
356     inet->mc_loop   = 1;
357     inet->mc_ttl    = 1;
358     inet->mc_all    = 1;
359     inet->mc_index  = 0;
360     inet->mc_list   = NULL;
361     inet->rcv_tos   = 0;
 …
376     if (sk->sk_prot->init) {
377         err = sk->sk_prot->init(sk);
 …
386 }
```

여기까지 따라옴

&tcp_prot

return to __sock_create()

__sock_create()

net/socket.c

```
1088 int __sock_create(struct net *net, int family, int type, int protocol,
1089                    struct socket **res, int kern)
1090 {
1091     int err;
1092     struct socket *sock;
1093     const struct net_proto_family *pf;
 …
1127     sock = sock_alloc();
 …
1134     sock->type = type;
 …
1148     pf = rcu_dereference(net_families[family]);
 …
1163     err = pf->create(net, sock, protocol, kern);
 …
1182     *res = sock;
 …
1198 }
```

PF_INET

SOCK_STREAM

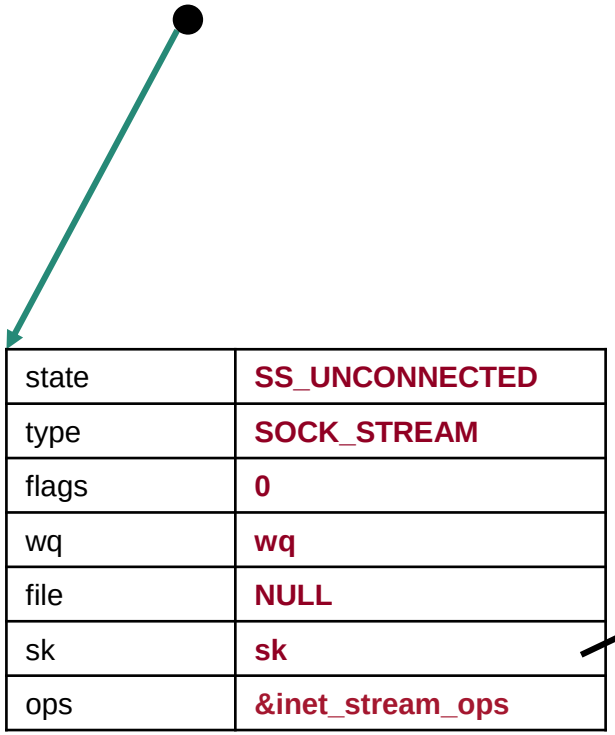
0

앞서 선언한 소켓

여기까지 따라옴

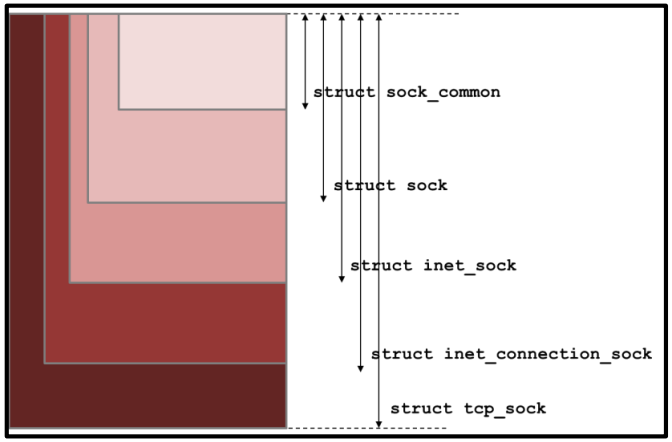
Up to now

sys_socketcall()
struct socket *sock



state	SS_UNCONNECTED
type	SOCK_STREAM
flags	0
wq	wq
file	NULL
sk	sk
ops	&inet_stream_ops

struct tcp_sock



res는 socket() 시스템콜에서 선언한 socket구조체의 포인터이다

return to sys_socketcall()

- socket(PF_INET, SOCK_STREAM, 0)

sys_socketcall()

net/socket.c

```
1213 SYSCALL_DEFINE3(socket, int, family, int, type, int, protocol)
1214 {
1215     int retval;
1216     struct socket *sock;
1217     int flags;
 …
1233     retval = sock_create(family, type, protocol, &sock);
1234     if (retval < 0)
1235         goto out;
1236
1237     retval = sock_map_fd(sock, flags & (O_CLOEXEC | O_NONBLOCK));
1238     if (retval < 0)
1239         goto out_release;
 …
```

여기까지 따라옴

소켓과 관련된 파일시스템 구조체를 생성, 초기화

sock_map_fd()

sock_map_fd()

net/socket.c

```
391 static int sock_map_fd(struct socket *sock, int flags)
392 {
393     struct file *newfile;
394     int fd = get_unused_fd_flags(flags);
 …
398     newfile = sock_alloc_file(sock, flags, NULL);
399     if (likely(!IS_ERR(newfile))) {
400         fd_install(fd, newfile);
401         return fd;
402     }
 …
406 }
407
```

새로운 file 구조체 allocation, fd값을 리턴

sock_alloc_file()

sock_alloc_file()

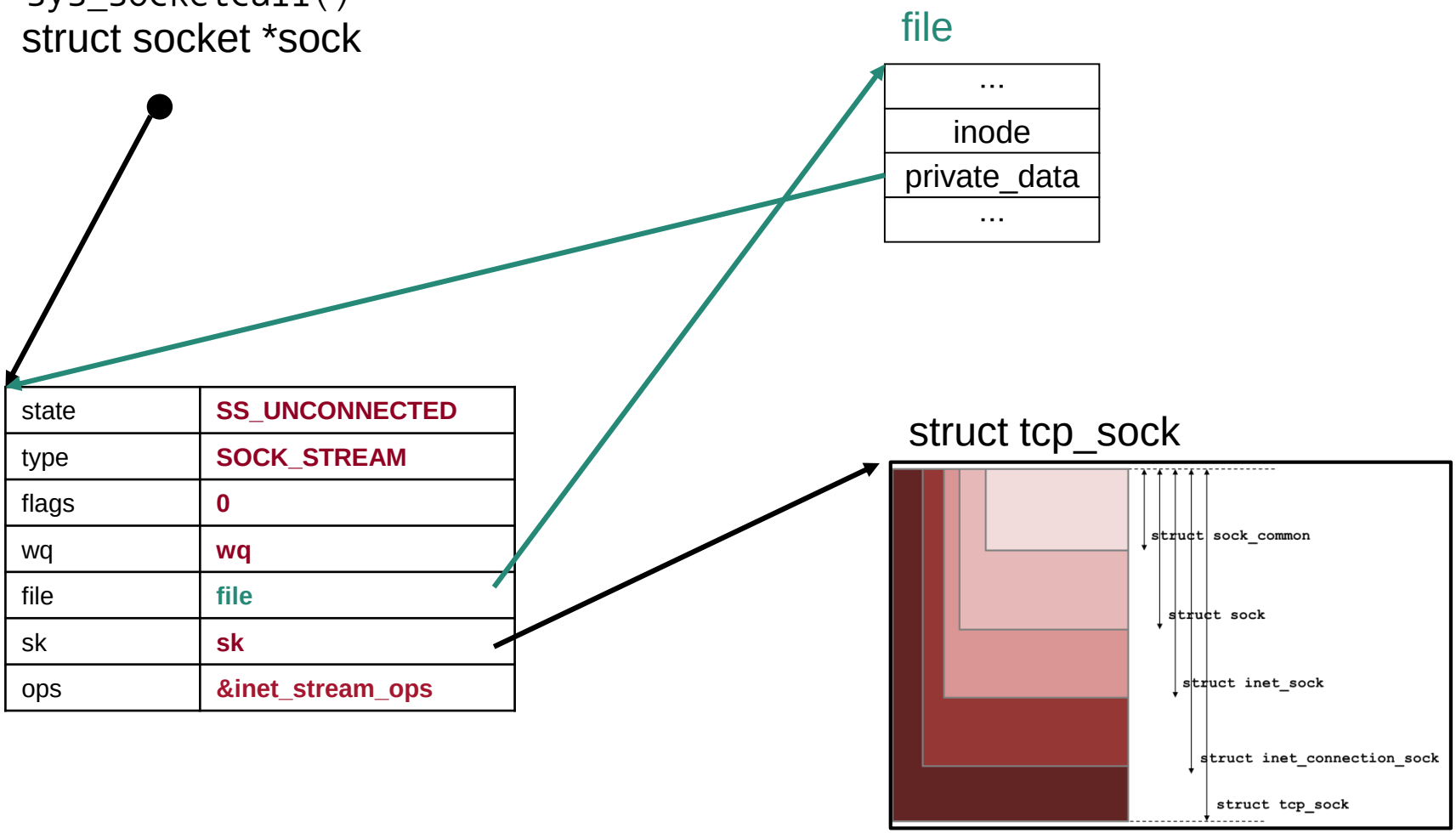
net/socket.c

```
355 struct file *sock_alloc_file(struct socket *sock, int flags, const char *dname)
356 {
 …
359         struct file *file;
 …
375         file = alloc_file(&path, FMODE_READ | FMODE_WRITE,
376                   &socket_file_ops);
 …
384         sock->file = file;
385         file->f_flags = O_RDWR | (flags & O_NONBLOCK);
386         file->private_data = sock;
387         return file;
388 }
```

file 구조체 allocation -> file의 f_op필드는 socket_file_ops
socket의 file포인터와 file의 private_data 포인터를 서로 할당

Up to now

sys_socketcall()
struct socket *sock



sock_map_fd()에서 struct file을 생성하고 해당 포인터에 주소 할당 및 fd설치

sock_map_fd()

sock_map_fd()

net/socket.c

```
391 static int sock_map_fd(struct socket *sock, int flags)
392 {
393     struct file *newfile;
394     int fd = get_unused_fd_flags(flags);
 …
398     newfile = sock_alloc_file(sock, flags, NULL);
399     if (likely(!IS_ERR(newfile))) {
400         fd_install(fd, newfile);
401         return fd;
402     }
 …
406 }
407
```

새로운 file 구조체 allocation, fd값을 리턴