Document d'Exigences Fonctionnelles
Stack Réseau TCP/IP + API Socket en C

---

1. INTRODUCTION

1.1 Objet
Ce document spécifie les exigences fonctionnelles pour l'implémentation d'une stack réseau TCP/IP complète avec une API socket en langage C.

1.2 Portée
- Implémentation des couches Réseau (IP), Transport (TCP/UDP)
- API socket compatible BSD/POSIX
- Gestion statique des ressources (tables, buffers)
- Architecture mono-thread avec multiplexage I/O

1.3 Architecture en couches
```
┌─────────────────────────────────────────┐
│    Application          │
├─────────────────────────────────────────┤
│    API Socket           │ ← Interface applicative
├─────────────────────────────────────────┤
│  Couche Transport (TCP/UDP) │ ← Fiabilité, multiplexage
├─────────────────────────────────────────┤
│  Couche Réseau (IP)       │ ← Routage, fragmentation
├─────────────────────────────────────────┤
│  Couche Liaison (Ethernet) │ ← Interface matérielle
└─────────────────────────────────────────┘
```

---

2. COUCHE LIAISON DE DONNÉES (Interface Ethernet)

2.1 Interface avec le matériel

EXG-ETH-001: Interface Raw Socket
- Création d'un raw socket (AF_PACKET sur Linux, BPF sur BSD)
- Accès direct aux trames Ethernet
- Mode promiscuité si nécessaire

EXG-ETH-002: Envoi de trames Ethernet
```c
int eth_send_frame(const uint8_t *dst_mac, const uint8_t *src_mac,
                   uint16_t ethertype, const void *payload, size_t len);
```
- Construction de l'en-tête Ethernet (14 octets)
- Support Ethernet II uniquement
- Ethertype: 0x0800 (IPv4), 0x0806 (ARP)

EXG-ETH-003: Réception de trames Ethernet
```c
int eth_recv_frame(uint8_t *frame_buffer, size_t max_len,
                   struct eth_header *eth_hdr);
```
- Lecture depuis le raw socket
- Parsing de l'en-tête Ethernet
- Dispatch selon l'ethertype (ARP ou IP)

2.2 Gestion des adresses MAC

EXG-ETH-010: Table ARP statique
```c
#define ARP_TABLE_SIZE 128

typedef struct {
    uint32_t ip_addr;          // Adresse IPv4 (network byte order)
    uint8_t mac_addr[6];       // Adresse MAC
    time_t timestamp;          // Date de dernière mise à jour
    bool is_valid;             // Entrée valide ou libre
} arp_entry_t;

static arp_entry_t g_arp_table[ARP_TABLE_SIZE];
```

EXG-ETH-011: Résolution ARP
```c
int arp_resolve(uint32_t ip_addr, uint8_t mac_addr[6]);
```
- Recherche dans la table ARP locale
- Envoi d'une requête ARP (broadcast) si non trouvé
- Attente de la réponse ARP avec timeout
- Mise à jour de la table

EXG-ETH-012: Traitement des paquets ARP
```c
int arp_handle_packet(const uint8_t *arp_packet, size_t len);
```
- Traitement des requêtes ARP (who-has)
- Envoi de réponses ARP si IP locale
- Traitement des réponses ARP (is-at)
- Mise à jour de la table ARP

EXG-ETH-013: Expiration des entrées ARP
- Timeout par défaut: 300 secondes
- Fonction `arp_cleanup()` appelée périodiquement
- Suppression des entrées expirées

---

3. COUCHE RÉSEAU (IP)

3.1 Configuration IP

EXG-IP-001: Configuration de l'interface
```c
typedef struct {
    uint32_t ip_addr;          // Adresse IP locale
    uint32_t netmask;          // Masque de sous-réseau
    uint32_t gateway;          // Passerelle par défaut
    uint8_t mac_addr[6];       // Adresse MAC locale
} ip_config_t;

int ip_configure(const ip_config_t *config);
```

EXG-IP-002: Validation de configuration
- Vérification de la cohérence IP/masque
- Calcul automatique de l'adresse réseau
- Calcul de l'adresse de broadcast

3.2 Traitement des paquets IP

EXG-IP-010: Réception de paquets IP
```c
int ip_recv_packet(uint8_t *packet_buffer, size_t max_len,
                   struct ip_header *ip_hdr);
```

```
- Validation de l'en-tête IP (version, IHL, checksum)
- Vérification de l'adresse IP destination (unicast/broadcast)
- Défragmentation si nécessaire
- Dispatch selon le protocole (ICMP, TCP, UDP)

**EXG-IP-011**: Structure de l'en-tête IPv4
```c
typedef struct {
    uint8_t version_ihl;       // Version (4 bits) + IHL (4 bits)
    uint8_t tos;               // Type of Service
    uint16_t total_length;     // Longueur totale
    uint16_t identification;   // ID pour fragmentation
    uint16_t flags_offset;     // Flags (3 bits) + Fragment Offset (13 bits)
    uint8_t ttl;               // Time To Live
    uint8_t protocol;          // IPPROTO_ICMP, TCP, UDP
    uint16_t header_checksum;  // Checksum de l'en-tête
    uint32_t src_addr;         // Adresse IP source
    uint32_t dst_addr;         // Adresse IP destination
} __attribute__((packed)) ip_header_t;
```

EXG-IP-012: Envoi de paquets IP
```c
int ip_send_packet(uint32_t dst_ip, uint8_t protocol,
                   const void *payload, size_t len, uint8_t ttl);
```
- Construction de l'en-tête IP
- Calcul du checksum de l'en-tête
- Fragmentation si payload > MTU (1500 octets)
- Résolution ARP de la destination
- Envoi via la couche Ethernet

EXG-IP-013: Calcul du checksum IP
```c
uint16_t ip_checksum(const void *data, size_t len);
```
- Algorithme standard RFC 1071
- Complément à 1 de la somme en complément à 1 des mots de 16 bits

3.3 Fragmentation et réassemblage

EXG-IP-020: Fragmentation à l'envoi
```

- Détection si payload > MTU
- Génération d'un ID unique pour le datagramme
- Création de fragments avec offsets corrects
- Flag MF (More Fragments) sauf pour le dernier
- Chaque fragment a sa propre en-tête IP

EXG-IP-021: Table de réassemblage
```c
#define MAX_FRAGMENTS 16
#define MAX_REASSEMBLY_ENTRIES 32

typedef struct {
    uint32_t src_ip;
    uint32_t dst_ip;
    uint16_t identification;
    uint8_t protocol;
    uint8_t fragments[MAX_FRAGMENTS];  // Bitmap des fragments reçus
    uint8_t *data;                     // Buffer de réassemblage
    size_t total_length;
    time_t first_fragment_time;
    bool complete;
} reassembly_entry_t;

static reassembly_entry_t g_reassembly_table[MAX_REASSEMBLY_ENTRIES];
```

EXG-IP-022: Réassemblage à la réception
- Identification des fragments (src, dst, ID, protocole)
- Allocation d'une entrée de réassemblage
- Copie du fragment à l'offset correct
- Détection du dernier fragment (MF=0)
- Livraison du datagramme complet
- Timeout de réassemblage: 30 secondes

3.4 Protocole ICMP

EXG-IP-030: Support ICMP minimal
```c
typedef struct {
    uint8_t type;
    uint8_t code;
    uint16_t checksum;
    union {
```

```
        struct {
            uint16_t id;
            uint16_t sequence;
        } echo;
        uint32_t gateway;
    } data;
} __attribute__((packed)) icmp_header_t;
```

EXG-IP-031: Traitement ICMP Echo Request (ping)
- Réception d'un Echo Request (type 8)
- Génération d'un Echo Reply (type 0)
- Copie du payload original
- Envoi de la réponse

EXG-IP-032: Messages ICMP d'erreur
- Destination Unreachable (type 3)
- Time Exceeded (type 11)
- Génération lors d'erreurs de routage ou TTL=0

---

4. COUCHE TRANSPORT - UDP

4.1 Structure des segments UDP

EXG-UDP-001: En-tête UDP
```c
typedef struct {
    uint16_t src_port;
    uint16_t dst_port;
    uint16_t length;           // Longueur en-tête + données
    uint16_t checksum;         // Optionnel en IPv4
} __attribute__((packed)) udp_header_t;
```

EXG-UDP-002: Calcul du checksum UDP
- Utilisation d'un pseudo-en-tête IP
- Pseudo-en-tête: src_ip, dst_ip, protocol, udp_length
- Checksum calculé sur pseudo-en-tête + en-tête UDP + données

4.2 Table des sockets UDP

EXG-UDP-010: Table de binding UDP
```c
#define MAX_UDP_SOCKETS 64

typedef struct {
    int socket_index;          // Index dans la table socket globale
    uint16_t local_port;       // Port local (network byte order)
    uint32_t local_ip;         // IP locale (0 = any)
    uint16_t remote_port;      // Port distant (0 si non connecté)
    uint32_t remote_ip;        // IP distante (0 si non connecté)
    bool is_bound;
    bool is_connected;
} udp_socket_t;

static udp_socket_t g_udp_sockets[MAX_UDP_SOCKETS];
```

EXG-UDP-011: Allocation de port UDP
```c
uint16_t udp_alloc_ephemeral_port(void);
```
- Ports éphémères: 49152-65535
- Recherche d'un port libre dans la table
- Incrémentation cyclique du dernier port utilisé

4.3 Opérations UDP

EXG-UDP-020: Bind UDP
```c
int udp_bind(int socket_idx, uint16_t port);
```
- Vérification que le port n'est pas déjà utilisé
- Association socket_idx ↔ port dans la table UDP
- Permet de recevoir sur ce port

EXG-UDP-021: Connect UDP (optionnel)
```c
int udp_connect(int socket_idx, uint32_t dst_ip, uint16_t dst_port);
```
- Stockage de l'adresse/port distant
- Les futurs send() utiliseront ces valeurs par défaut
- Filtrage des réceptions (seul le peer connecté)

EXG-UDP-022: Envoi UDP
```c
int udp_sendto(int socket_idx, const void *data, size_t len,
               uint32_t dst_ip, uint16_t dst_port);
```
- Allocation d'un port local si pas encore bound
- Construction de l'en-tête UDP
- Calcul du checksum UDP
- Appel de ip_send_packet() avec protocole IPPROTO_UDP

EXG-UDP-023: Réception UDP
```c
int udp_recvfrom(int socket_idx, void *buffer, size_t max_len,
                 uint32_t *src_ip, uint16_t *src_port);
```
- Consultation de la file d'attente de réception du socket
- Copie des données vers le buffer utilisateur
- Retourne l'adresse source

4.4 Files de réception UDP

EXG-UDP-030: Queue de datagrammes UDP
```c
#define UDP_RECV_QUEUE_SIZE 16

typedef struct {
    uint8_t data[2048];        // Données du datagramme
    size_t len;
    uint32_t src_ip;
    uint16_t src_port;
} udp_datagram_t;

typedef struct {
    udp_datagram_t queue[UDP_RECV_QUEUE_SIZE];
    int head;                  // Index d'écriture
    int tail;                  // Index de lecture
    int count;                 // Nombre d'éléments
} udp_recv_queue_t;
```

EXG-UDP-031: Enqueue de datagrammes
- Fonction appelée lors de la réception d'un paquet UDP
- Recherche du socket destinataire par port

- Ajout dans la queue circulaire
- Rejet si queue pleine (datagramme perdu)

EXG-UDP-032: Dequeue de datagrammes
- Fonction appelée lors d'un recvfrom()
- Extraction du premier datagramme
- Retourne EAGAIN si queue vide (mode non-bloquant)

---

5. COUCHE TRANSPORT - TCP

5.1 Structure des segments TCP

EXG-TCP-001: En-tête TCP
```c
typedef struct {
    uint16_t src_port;
    uint16_t dst_port;
    uint32_t seq_num;          // Numéro de séquence
    uint32_t ack_num;          // Numéro d'acquittement
    uint8_t data_offset_flags;  // Data Offset (4 bits) + Reserved (4 bits)
    uint8_t flags;             // CWR, ECE, URG, ACK, PSH, RST, SYN, FIN
    uint16_t window_size;      // Fenêtre de réception
    uint16_t checksum;
    uint16_t urgent_pointer;
} __attribute__((packed)) tcp_header_t;

// Flags TCP
#define TCP_FIN  0x01
#define TCP_SYN  0x02
#define TCP_RST  0x04
#define TCP_PSH  0x08
#define TCP_ACK  0x10
#define TCP_URG  0x20
```

EXG-TCP-002: Calcul du checksum TCP
- Même principe que UDP avec pseudo-en-tête IP
- Checksum obligatoire pour TCP

5.2 Machine à états TCP

EXG-TCP-010: États TCP
```c
typedef enum {
    TCP_STATE_CLOSED,
    TCP_STATE_LISTEN,
    TCP_STATE_SYN_SENT,
    TCP_STATE_SYN_RECEIVED,
    TCP_STATE_ESTABLISHED,
    TCP_STATE_FIN_WAIT_1,
    TCP_STATE_FIN_WAIT_2,
    TCP_STATE_CLOSE_WAIT,
    TCP_STATE_CLOSING,
    TCP_STATE_LAST_ACK,
    TCP_STATE_TIME_WAIT
} tcp_state_t;
```

EXG-TCP-011: Diagramme de transitions d'états
```
CLOSED → LISTEN (passive open)
CLOSED → SYN_SENT (active open) → ESTABLISHED
LISTEN → SYN_RECEIVED → ESTABLISHED

ESTABLISHED → FIN_WAIT_1 → FIN_WAIT_2 → TIME_WAIT → CLOSED
ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED
```

5.3 Bloc de contrôle TCP (TCB)

EXG-TCP-020: Structure TCB
```c
#define MAX_TCP_SOCKETS 64

typedef struct {
    int socket_index;          // Index dans table socket globale
    tcp_state_t state;

    // Adresses
    uint32_t local_ip;
    uint16_t local_port;
    uint32_t remote_ip;
    uint16_t remote_port;
```

```
    // Numéros de séquence
    uint32_t snd_una;         // Send Unacknowledged
    uint32_t snd_nxt;         // Send Next
    uint32_t snd_wnd;         // Send Window
    uint32_t snd_iss;         // Initial Send Sequence

    uint32_t rcv_nxt;         // Receive Next
    uint32_t rcv_wnd;         // Receive Window
    uint32_t rcv_irs;         // Initial Receive Sequence

    // Buffers
    uint8_t send_buffer[8192];
    size_t send_buffer_used;
    uint8_t recv_buffer[8192];
    size_t recv_buffer_used;

    // Retransmission
    time_t last_send_time;
    uint32_t rto;             // Retransmission Timeout (ms)
    uint8_t retransmit_count;

    // Timers
    time_t time_wait_start;

} tcp_control_block_t;

static tcp_control_block_t g_tcp_sockets[MAX_TCP_SOCKETS];
```

5.4 Connexion TCP (3-way handshake)

EXG-TCP-030: Active Open (Client)
```c
int tcp_connect(int socket_idx, uint32_t dst_ip, uint16_t dst_port);
```
1. Allocation d'un port local éphémère
2. Génération d'un ISN (Initial Sequence Number) aléatoire
3. Envoi d'un segment SYN (seq=ISN)
4. Transition vers SYN_SENT
5. Attente du SYN+ACK
6. Envoi d'un ACK
7. Transition vers ESTABLISHED

EXG-TCP-031: Passive Open (Serveur)
```c
int tcp_listen(int socket_idx, int backlog);
```
1. Socket passe en état LISTEN
2. Backlog définit la queue de connexions en attente
3. À la réception d'un SYN:
   - Allocation d'un nouveau TCB
   - Envoi d'un SYN+ACK
   - Transition vers SYN_RECEIVED
4. À la réception du ACK final:
   - Transition vers ESTABLISHED
   - Ajout à la queue d'accept

EXG-TCP-032: Accept de connexion
```c
int tcp_accept(int listen_socket_idx, uint32_t *peer_ip, uint16_t *peer_port);
```
- Retourne le socket_idx de la nouvelle connexion
- Extrait de la queue d'accept
- Bloque ou retourne EAGAIN si queue vide

5.5 Transfert de données TCP

EXG-TCP-040: Envoi de données
```c
int tcp_send(int socket_idx, const void *data, size_t len);
```
1. Vérification de l'état (doit être ESTABLISHED)
2. Copie des données dans le send_buffer
3. Construction du segment TCP avec les données
4. seq = snd_nxt
5. Envoi via IP
6. Mise à jour de snd_nxt += len
7. Démarrage du timer de retransmission
8. Les données restent dans le buffer jusqu'à l'ACK

EXG-TCP-041: Réception de données
```c
int tcp_recv(int socket_idx, void *buffer, size_t max_len);
```
1. Vérification de l'état
2. Copie des données depuis recv_buffer vers buffer utilisateur

3. Retourne le nombre d'octets copiés
4. Mise à jour de la fenêtre de réception

EXG-TCP-042: Traitement des segments entrants
```c
void tcp_handle_segment(const tcp_header_t *tcp_hdr, const uint8_t *data,
                        size_t data_len, uint32_t src_ip, uint32_t dst_ip);
```
1. Calcul et validation du checksum
2. Recherche du TCB correspondant (quadruplet)
3. Traitement selon l'état:
   - Vérification des numéros de séquence
   - Traitement des flags (SYN, ACK, FIN, RST)
   - Extraction des données
   - Envoi des ACK
   - Transitions d'état

5.6 Acquittements TCP

EXG-TCP-050: Envoi d'ACK
- ACK envoyés pour chaque segment de données reçu (ou delayed ACK)
- ack_num = rcv_nxt (prochain octet attendu)
- Mise à jour de la fenêtre

EXG-TCP-051: Traitement des ACK reçus
- Validation: snd_una < ack_num ≤ snd_nxt
- Suppression des données acquittées du send_buffer
- Mise à jour de snd_una
- Annulation du timer de retransmission si tout est ACK
- Mise à jour de la fenêtre d'envoi (snd_wnd)

EXG-TCP-052: Retransmission
- Timer RTO (Retransmission Timeout) par défaut: 1 seconde
- Si timeout expire sans ACK, retransmission du segment
- Doublement du RTO (exponential backoff)
- Maximum 5 retransmissions puis RST

5.7 Contrôle de flux

EXG-TCP-060: Fenêtre de réception
- rcv_wnd annoncée dans chaque segment envoyé
- rcv_wnd = taille_buffer - données_en_attente
- L'émetteur ne doit pas dépasser la fenêtre annoncée

EXG-TCP-061: Fenêtre d'envoi
- snd_wnd = fenêtre annoncée par le récepteur
- Limitation de la quantité de données envoyées mais non acquittées
- snd_nxt - snd_una ≤ snd_wnd

EXG-TCP-062: Gestion des buffers
- Buffer d'envoi: données non encore acquittées
- Buffer de réception: données reçues mais non lues par l'application
- Détection de buffer plein et signalement (fenêtre à 0)

5.8 Fermeture de connexion

EXG-TCP-070: Fermeture active (close)
```c
int tcp_close(int socket_idx);
```
1. État ESTABLISHED → envoi FIN → FIN_WAIT_1
2. Réception ACK → FIN_WAIT_2
3. Réception FIN → envoi ACK → TIME_WAIT
4. Attente 2*MSL (120 secondes) → CLOSED

EXG-TCP-071: Fermeture passive
1. État ESTABLISHED, réception FIN → envoi ACK → CLOSE_WAIT
2. Application appelle close() → envoi FIN → LAST_ACK
3. Réception ACK → CLOSED

EXG-TCP-072: Fermeture simultanée
1. État ESTABLISHED → envoi FIN → FIN_WAIT_1
2. Réception FIN → envoi ACK → CLOSING
3. Réception ACK du FIN → TIME_WAIT → CLOSED

EXG-TCP-073: Reset de connexion (RST)
- Envoi de RST en cas d'erreur ou port fermé
- Réception de RST → fermeture immédiate, passage à CLOSED
- Pas de TIME_WAIT en cas de RST

---

6. API SOCKET

6.1 Table de sockets globale

EXG-SOCK-001: Table unifiée TCP/UDP

```c
#define MAX_SOCKETS 128

typedef enum {
    SOCKET_TYPE_NONE,
    SOCKET_TYPE_TCP,
    SOCKET_TYPE_UDP,
    SOCKET_TYPE_RAW
} socket_type_t;

typedef enum {
    SOCKET_STATE_FREE,
    SOCKET_STATE_CREATED,
    SOCKET_STATE_BOUND,
    SOCKET_STATE_LISTENING,
    SOCKET_STATE_CONNECTING,
    SOCKET_STATE_CONNECTED,
    SOCKET_STATE_CLOSING,
    SOCKET_STATE_CLOSED
} socket_state_t;

typedef struct {
    socket_type_t type;
    socket_state_t state;
    int domain;              // AF_INET
    int protocol;            // IPPROTO_TCP, IPPROTO_UDP

    union {
        int tcp_index;       // Index dans g_tcp_sockets
        int udp_index;       // Index dans g_udp_sockets
    };

    // Options socket
    bool non_blocking;
    bool reuse_addr;
    struct timeval rcv_timeout;
    struct timeval snd_timeout;

    // Statistiques
    time_t created_at;
    uint64_t bytes_sent;
    uint64_t bytes_received;
```

```
} socket_entry_t;

static socket_entry_t g_socket_table[MAX_SOCKETS];
```

6.2 Création et configuration

EXG-SOCK-010: socket()
```c
int socket(int domain, int type, int protocol);
```
- domain: AF_INET uniquement
- type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP), SOCK_RAW
- protocol: 0 (auto), IPPROTO_TCP, IPPROTO_UDP
- Retourne un descripteur de socket (index dans g_socket_table)
- Allocation d'un TCB ou UDP socket selon le type

EXG-SOCK-011: setsockopt()
```c
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```
Options supportées:
- SOL_SOCKET / SO_REUSEADDR: réutilisation d'adresse
- SOL_SOCKET / SO_RCVTIMEO: timeout de réception
- SOL_SOCKET / SO_SNDTIMEO: timeout d'envoi
- IPPROTO_TCP / TCP_NODELAY: désactivation de Nagle (optionnel)

EXG-SOCK-012: getsockopt()
```c
int getsockopt(int sockfd, int level, int optname,
               void *optval, socklen_t *optlen);
```
- Lecture des options configurées
- Lecture de statistiques (SO_ERROR, etc.)

EXG-SOCK-013: fcntl() - Mode non-bloquant
```c
int fcntl(int sockfd, int cmd, ...);
```
- F_SETFL avec O_NONBLOCK
- Les opérations retournent EAGAIN/EWOULDBLOCK si non prêtes

6.3 Opérations de binding et écoute

EXG-SOCK-020: bind()
```c
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```
- addr: struct sockaddr_in avec IP et port
- Port 0 = allocation automatique d'un port éphémère
- Appel de tcp_bind() ou udp_bind() selon le type
- Vérification qu'aucun autre socket n'utilise ce port

EXG-SOCK-021: listen()
```c
int listen(int sockfd, int backlog);
```
- Uniquement pour SOCK_STREAM (TCP)
- backlog: taille de la queue de connexions en attente
- Transition vers SOCKET_STATE_LISTENING
- Appel de tcp_listen()

EXG-SOCK-022: accept()
```c
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```
- Extraction d'une connexion de la queue d'accept
- Retourne un nouveau descripteur de socket
- addr: informations du peer (IP, port)
- Bloquant ou non selon le mode du socket

6.4 Connexion

EXG-SOCK-030: connect()
```c
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```
- TCP: initie le 3-way handshake via tcp_connect()
- UDP: enregistre l'adresse distante (connected UDP)
- Mode bloquant: attend la fin du handshake
- Mode non-bloquant: retourne EINPROGRESS immédiatement

6.5 Transfert de données

EXG-SOCK-040: send()
```c
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```
- TCP: appelle tcp_send()
- UDP connecté: appelle udp_sendto() avec adresse mémorisée
- Retourne le nombre d'octets envoyés (peut être < len)
- Flags: MSG_DONTWAIT, MSG_NOSIGNAL (optionnels)

EXG-SOCK-041: recv()
```c
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```
- TCP: appelle tcp_recv()
- UDP connecté: appelle udp_recvfrom() avec filtrage peer
- Retourne 0 si connexion fermée (TCP)
- Retourne -1/EAGAIN si pas de données (non-bloquant)

EXG-SOCK-042: sendto()
```c
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```
- Principalement pour UDP
- Permet de spécifier la destination pour chaque envoi
- Appelle udp_sendto()

EXG-SOCK-043: recvfrom()
```c
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```