🔹 `socket()`

c

```
int socket(int domain, int type, int protocol);
```

Tworzy nowy **gniazdo (socket)** i zwraca jego deskryptor pliku.

- `domain`: rodzina protokołu, np. `AF_INET` (IPv4), `AF_UNIX` (lokalne)
- `type`: typ gniazda, np. `SOCK_STREAM` (TCP), `SOCK_DGRAM` (UDP)
- `protocol`: zazwyczaj 0 – system sam wybierze odpowiedni

Zwraca: deskryptor socketu lub `-1` jeśli błąd.

🔹 `make_local_socket()`

To funkcja własna w Twoim kodzie (`l8_common.h`), która:

1. Wywołuje `socket(AF_UNIX, SOCK_STREAM, 0)`
2. Wypełnia strukturę `sockaddr_un` nazwą pliku (gniazda lokalnego)
3. Zwraca deskryptor socketu

🔹 `connect()`

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

Łączy socket (klienta) z podanym adresem (serwera).

- `sockfd`: deskryptor socketu
- `addr`: wskaźnik na strukturę `sockaddr_in` lub `sockaddr_un`
- `addrlen`: długość struktury

Zwraca: 0 jeśli sukces, `-1` jeśli błąd.

🔹 unlink()

```
int unlink(const char *pathname);
```

Usuwa plik z systemu plików. Używane w Twoim serwerze do **usunięcia starego gniazda lokalnego** (`AF_UNIX`).

🔹 bind()

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

Przypisuje adres do socketu serwera, np. port TCP lub ścieżkę do pliku lokalnego.

Używane przez serwer do powiązania socketu z konkretnym adresem IP/portem lub plikiem UNIX.

🔹 listen()

```
int listen(int sockfd, int backlog);
```

Ustawia socket w tryb nasłuchu – serwer jest gotowy na przyjmowanie połączeń (`accept()`).

- `backlog` – maksymalna liczba oczekujących połączeń

🔹 getaddrinfo()

```
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints, struct addrinfo
**res);
```

Zamienia adres IP/nazwę domenową i port na strukturę `sockaddr_in`.

- `node`: np. `"localhost"`
- `service`: np. `"2000"`
- `hints`: filtry, np. `AF_INET`, `SOCK_STREAM`
- `res`: wynik – wskaźnik na listę adresów

Używane w `make_address()` by stworzyć adres serwera TCP.

🌀 EPOLL – obsługa wielu klientów bez wątków

🔹 `epoll_create1()`

```
int epoll_create1(int flags);
```

Tworzy nową instancję epolla – strukturę do zarządzania wieloma gniazdami.

- `flags = 0` lub `EPOLL_CLOEXEC`
- Zwraca deskryptor epolla (np. `efd`), używany w `epoll_ctl` i `epoll_wait`.

🔹 `epoll_ctl()`

```
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
```

Dodaje/usuwa/modyfikuje `fd` (np. socket) w epollu.

- `epfd`: deskryptor epolla
- `op`: `EPOLL_CTL_ADD`, `EPOLL_CTL_MOD`, `EPOLL_CTL_DEL`
- `fd`: który socket kontrolować
- `event`: struktura mówiąca, jakie zdarzenia nas interesują (np. `EPOLLIN`)

🔹 `epoll_wait()` / `epoll_pwait()`

```
int epoll_wait(int epfd, struct epoll_event *events, int maxevents,
int timeout);
```

Czeka na zdarzenia na zarejestrowanych socketach.

- `events`: tablica struktur `epoll_event`, do której wpisane zostaną aktywne sockety
- `maxevents`: ile jednocześnie maksymalnie odebrać
- `timeout`: -1 = czeka w nieskończoność

Twoja wersja używa `epoll_pwait()` – jak wyżej, ale dodatkowo ustawia maskę sygnałów na czas czekania.

🔹 `add_new_client()`

To funkcja z `l8_common.h`:

```
int add_new_client(int sfd);
```

Używa `accept()`, by przyjąć nowe połączenie od klienta:

- `sfd` – socket nasłuchujący (lokalny lub TCP)
- Zwraca nowy deskryptor do komunikacji z klientem

⚙️ Inne funkcje systemowe

🔹 `fcntl()`

```
int fcntl(int fd, int cmd, ... /* arg */ );
```

Funkcja do zarządzania opcjami deskryptora pliku.

Najczęściej używana tak:

```
int flags = fcntl(fd, F_GETFL);
fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

- Ustawia **tryb nieblokujący** na socket – pozwala, by `read`/`accept` nie blokowały w epollu.