שימוש ב-UDS

עד כה, עבדנו ב-TCP , וב-UDP כדי לתקשר עם השרת דרך הרשת (זאת דרך IP ו← port).

כעת, בחלק זה נשתמש ב- **UDP ← Unix Domain Sockets** , זוהי דרך לתקשר בין תהליכים, אשר נמצאים על אותו מחשב, ולא דרך הרשת, אלא דרך קובץ במערכת הקבצים.

ישנה חשיבות להשתמש ב-UDS מאשר ב-UDP/TCP, זאת כי זה עובד רק באותו המחשב.

כעת, כמו בכל חלק , נעביר את כל הקבצים מכל חלק לחלק החדש שבו אנו עובדים.

נשדרג את השרת שלנו, ולכן נשנה את שמו ל← **uds_server** .

נוסיף לשרת המשודרג שלנו משתנים חדשים לשמירת הנתיבים של ה-UDS, זאת בשביל תמיכה ב← Unix Domain Sockets , הם עובדים עם קובץ path, שזהו קובץ unix, ולא עם IP או פורט:

```
//Variables for saving the pathes of the UDS:
std::string stream_path;
std::string datagram_path;
```

כמובן שגם נוסיף תמיכה בפרמטרים החדשים של שורת הפקודה:

s ← עבור ה stream , שזה כמו הTCP.

d ← עבור הdatagram, שזה כמו הUDP.

זאת על מנת שהמשתמש יוכל להריץ את השרת שלנו ככה:

./uds_server -s /tmp/uds_stream.sock -d /tmp/uds_dgram.sock

```
int opt;
while ((opt = getopt(argc, argv, "o:c:h:t:T:U:s:d:")) != -1) {
    switch (opt) {
        case 'o': oxygen_count = std::stoull(optarg); break;
        case 'c': carbon_count = std::stoull(optarg); break;
        case 'h': hydrogen_count = std::stoull(optarg); break;
        case 't': timeout_seconds = std::stoi(optarg); break;
        case 'T': tcp_port = std::stoi(optarg); break;
        case 'U': udp_port = std::stoi(optarg); break;
        case 's': stream_path = optarg; break;
        case 'd': datagram_path = optarg; break;
```

הפעם , ב-2 הקבצים הנוספים (מעבר לשרת שלנו) נבצע גם בהם שינויים על מנת שתהיינה תמיכה ב-UDS , נראה זאת בהמשך.

נמשיך בהסבר השרת, בנוסף לבדיקת תקינות ה-TCP, וה-UDP, נבדוק את קיום הארגומנטים, ונפעילם בהתאם:

```
if ((tcp_port == -1 || udp_port == -1) && stream_path.empty() && datagram_path.empty()) {
    std::cerr << "[ERROR] Must provide at least TCP/UDP or UDS stream/datagram options.\n";
    return 1;
}
```

כעת , ניצור את 2 ה-socketים המתקשרים ל-UDS :

```
// Create UDS STREAM socket
int uds_stream_fd = -1;
if (!stream_path.empty()) {
    uds_stream_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (uds_stream_fd < 0) {
        perror("UDS stream socket failed");
        return 1;
    }
```

כעת ניצור UDS socket מסוג **STREAM** בתקשורת המקומית AF_UNIX , נבדוק אם המשתמש סיפק נתיב s-, ל-UDS מסוג stream, במידה וכן נמשיך.

נבדוק תקינות מבחינת יצירת ה– socket (כמו שעשינו ב tcp/udp) .

נגדיר את מבנה הכתובת מסוג **sockaddr_un,** UNIX , ונאפס את הערכים שלנו , ונגדיר את הנתיב של הקובץ שייצג את ה-socket:

```
sockaddr_un stream_addr;
std::memset(&stream_addr, 0, sizeof(stream_addr));
stream_addr.sun_family = AF_UNIX;
std::strncpy(stream_addr.sun_path, stream_path.c_str(), sizeof(stream_addr.sun_path) - 1);
```

נמחק כל קובץ של socket קיים, זאת על מנת למנוע שגיאת bind() , ונבצע קישור בין ה-socket לקובץ במערכת הקבצים:

וכעת, הsocket שלנו מוכן כעת להאזין לחיבורים נכנסים:

```
unlink(stream_path.c_str()); // remove any existing socket file
if (bind(uds_stream_fd, (sockaddr*)&stream_addr, sizeof(stream_addr)) < 0) {
    perror("UDS stream bind failed");
    return 1;
}

if (listen(uds_stream_fd, 5) < 0) {
    perror("UDS stream listen failed");
    return 1;
}

std::cout << "[INFO] UDS STREAM socket bound to " << stream_path << "\n";
```

כעת, אנו עושים אותו הדבר עבור יצירת UDS socket מסוג **DATAGRAM,** נעשה כעת אותו הדבר בשלבים :

```
// Create UDS DATAGRAM socket
int uds_dgram_fd = -1;
if (!datagram_path.empty()) {
    uds_dgram_fd = socket(AF_UNIX, SOCK_DGRAM, 0);
    if (uds_dgram_fd < 0) {
        perror("UDS datagram socket failed");
        return 1;
    }

    sockaddr_un dgram_addr;
    std::memset(&dgram_addr, 0, sizeof(dgram_addr));
    dgram_addr.sun_family = AF_UNIX;
    std::strncpy(dgram_addr.sun_path, datagram_path.c_str(), sizeof(dgram_addr.sun_path) - 1);

    unlink(datagram_path.c_str());
    if (bind(uds_dgram_fd, (sockaddr*)&dgram_addr, sizeof(dgram_addr)) < 0) {
        perror("UDS datagram bind failed");
        return 1;
    }

    std::cout << "[INFO] UDS DATAGRAM socket bound to " << datagram_path << "\n";
}
```

כעת, נוסיף את ה-UDS STREAM וגם את ה- UDS DATAGRAM אל תוך ה-select שלנו, נוסיף אותו אל רשימת הקריאה **readfds** , כלומר← נבקש מה-select לבדוק אם מישהו מנסה להתחבר אלינו, או לשלוח לנו מידע דרך ה-socket הזה.

נבצע עדכון של ה-max_fd, שהוא המזהה הגבוה ביותר מבין כל ה-socketים שנבדקים← דרוש עבור הקריאה ל-select.

כל הדבר הזה קורה משום שאנחנו עובדים עם מספר סוגי תקשורת ← **UDS,UDP,TCP**
נצטרך שה-()select יעקוב אחרי כולם.

```
if (uds_stream_fd != -1) {
    FD_SET(uds_stream_fd, &readfds);
    if (uds_stream_fd > max_fd){
        max_fd = uds_stream_fd;
    }
}

if (uds_dgram_fd != -1) {
    FD_SET(uds_dgram_fd, &readfds);
    if (uds_dgram_fd > max_fd){
        max_fd = uds_dgram_fd;
    }
}
```

כעת, השרת שלנו מטפל גם בקבלת חיבורים חדשים דרך ה-UDS STREAM,

ודרך ה-UDS DATAGRAM , שום דבר לא השתנה כמו בבדיקת ה-tcp/udp:

```
if (uds_stream_fd != -1 && FD_ISSET(uds_stream_fd, &readfds)) {
    sockaddr_un client_addr;
    socklen_t client_len = sizeof(client_addr);
    int new_client = accept(uds_stream_fd, (sockaddr*)&client_addr, &client_len);
    if (new_client < 0) {
        perror("UDS stream accept failed");
    } else {
        clients.insert(new_client);
        std::cout << "[INFO] New UDS STREAM client connected.\n";
    }
}
```

```
if (uds_dgram_fd != -1 && FD_ISSET(uds_dgram_fd, &readfds)) {
    char buffer[1024] = {0};
    sockaddr_un client_addr;
    socklen_t client_len = sizeof(client_addr);
    ssize_t bytes_received = recvfrom(uds_dgram_fd, buffer, sizeof(buffer) - 1, 0,
                                      (sockaddr*)&client_addr, &client_len);
```

כעת, כאשר אני מסיים , אני רוצה לבצע ניקוי וסגירה מסודרת של ה-socketים בסיום ריצת השרת, עבור שני ה-socketים של ה-UDS:

הפעולה **unlink** הינה מוחקת את הקובץ של ה-socket מה-filesystem
(/tmp/my_stream.sock) , זאת כדי שהקובץ לא יישאר תקוע.

חובה לעשות unlink עבור UDS , אחרת הקובץ שלנו יישאר וימנע פתיחה מחודשת בהפעלה הבאה:

```
if (uds_stream_fd != -1) {
    close(uds_stream_fd);
    unlink(stream_path.c_str());
}

if (uds_dgram_fd != -1) {
    close(uds_dgram_fd);
    unlink(datagram_path.c_str());
}
```

כעת, הזכרנו למעלה כי גם בשני הקבצים **atom_supplier** ו-**molecule_requester** נבצע שינויים על מנת לתמוך ב-UDS , כעת נסביר מה הוספנו בכל קובץ:

עבור **atom_supplier**:

כעת , בניתוח הפרמטרים , אנו נוסיף תמיכה ב-UDS, זאת דרך f-

זה מאפשר להשתמש באופציה f/tmp/mysocket.sock כדי לעבוד עם UDS במקום TCP .

אם f- ניתן , אזי ה← uds_path יתמלא בשם קובץ הsocket במערכת הקבצים.

אנו קובעים האם להשתמש ב-UDS, או ב-TCP

```
std::string host;
int port=-1;
std::string uds_path;
int opt;

while((opt=getopt(argc,argv,"f:")) != -1){
    switch (opt){
        case 'f':
            uds_path=optarg;
            break;

        default:
            std::cerr << "Usage: " << argv[0] << " <host> <port> | -f <uds_path>\n";
            return 1;
    }
}

bool use_uds = !uds_path.empty();
if (!use_uds) {
    if (optind + 2 > argc) {
        std::cerr << "Usage: " << argv[0] << " <host> <port> | -f <uds_path>\n";
        return 1;
    }
    host = argv[optind];
    port = std::stoi(argv[optind + 1]);
}
```

במידה ונבצע התחברות לשרת UDS, אזי ניצור socket מסוג AF_UNIX עם SOCK_STREAM , נתחבר אל השרת דרך הקובץ שנמסר, במידה וזה מצליח← מתקבל

חיבור דו-כיווני עם השרת, אחרת נבצע התחברות רגילה דרך TCP .

```
int sock;
if (use_uds) {
    sock = socket(AF_UNIX, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("socket failed");
        return 1;
    }

    sockaddr_un addr{};
    addr.sun_family = AF_UNIX;
    std::strncpy(addr.sun_path, uds_path.c_str(), sizeof(addr.sun_path) - 1);

    if (connect(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("connect failed");
        return 1;
    }

    std::cout << "Connected to server via UDS at " << uds_path << "\n";
}
```

עבור **molecule_requester**:

גם כאן נצבע את הניתוח פרמטרים מהטרמינל ← אם ניתן הדגל f- אזי נשמר התניב לקובץ ה-socket המקומי, אחרת מצפים לפרמטרים רגילים , שזה -port, host :

```
std::string host;
int port = -1;
std::string uds_path;
int opt;

while ((opt = getopt(argc, argv, "f:")) != -1) {
    switch (opt) {
        case 'f':
            uds_path = optarg;
            break;
        default:
            std::cerr << "Usage: " << argv[0] << " <host> <port> | -f <uds_path>\n";
            return 1;
    }
}
```

ניצור את ה-UDS socket ונבצע bind לכתובת זמנית, מצבעים שימוש ב-**getpid()** כדי ליצור שם ייחודי, ולא להתנגש עם לקוחות אחרים:

```
// Step 1: Bind to a temporary client path so server can reply
sockaddr_un client_addr{};
client_addr.sun_family = AF_UNIX;

// Unique client socket path using the process ID
client_path = "/tmp/molecule_requester_" + std::to_string(getpid()) + ".sock";
std::strncpy(client_addr.sun_path, client_path.c_str(), sizeof(client_addr.sun_path) - 1);

// Make sure the path doesn't already exist
unlink(client_path.c_str());

if (bind(sock, (sockaddr*)&client_addr, sizeof(client_addr)) < 0) {
    perror("bind failed");
    return 1;
}
```

נבצע סגירה מסודרת בסיום, אם מדובר ב-UDS ← נמחק את הקובץ של ה-socket המקומי שפתחנו עם ה-()bind:

```
if (use_uds) {
    unlink(client_path.c_str()); // Cleanup the temporary UDS client socket
}
```

כעת נבצע הרצה מסודרת של הכל:

נבצע הידור מלא עם ה-**makefile** שברשותנו :

ונריץ את השרת שלנו, עם כל הפרוטוקולים+timeout :

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules$ cd Uds
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ make
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -o uds_server uds_server.cpp -lgcov
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -o molecule_requester molecule_requester.
cpp -lgcov
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -o atom_supplier atom_supplier.cpp -lgcov
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./uds_server -T 5555 -U 6666 \
-s /tmp/uds_stream.sock \
-d /tmp/uds_dgram.sock \
-h 30 -o 30 -c 30 -t 60
[INFO] UDS STREAM socket bound to /tmp/uds_stream.sock
[INFO] UDS DATAGRAM socket bound to /tmp/uds_dgram.sock
Server is listening on TCP port 5555 and UDP port 6666...
```

כעת נריץ את ה-atom_supplier:

נריץ ראשית לקוח TCP רגיל ואז נריץ לקוח UDS מסוג stream:

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules$ cd Uds
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./atom_supplier 127.0.0.1 5555
Connected to server at 127.0.0.1:5555
Enter command (e.g., ADD HYDROGEN 3 or EXIT): ADD HYDROGEN 5
Server returned: SUCCESS: Atom added successfully.

Enter command (e.g., ADD HYDROGEN 3 or EXIT): ADD OXYGEN 4
Server returned: SUCCESS: Atom added successfully.

Enter command (e.g., ADD HYDROGEN 3 or EXIT): EXIT
Closing connection. Bye!
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$
```

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules$ cd Uds
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./atom_supplier -f /tmp/uds_stream.sock
Connected to server via UDS at /tmp/uds_stream.sock
Enter command (e.g., ADD HYDROGEN 3 or EXIT): ADD HYDROGEN 5
Server returned: SUCCESS: Atom added successfully.

Enter command (e.g., ADD HYDROGEN 3 or EXIT): ADD OXYGEN 4
Server returned: SUCCESS: Atom added successfully.

Enter command (e.g., ADD HYDROGEN 3 or EXIT): EXIT
Closing connection. Bye!
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$
```

כעת התנהגות השרת שלנו עד כה:

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./uds_server -T 5555 -U 6666 \
-s /tmp/uds_stream.sock \
-d /tmp/uds_dgram.sock \
-h 30 -o 30 -c 30 -t 60
[INFO] UDS STREAM socket bound to /tmp/uds_stream.sock
[INFO] UDS DATAGRAM socket bound to /tmp/uds_dgram.sock
Server is listening on TCP port 5555 and UDP port 6666...
[INFO] New client connected: 127.0.0.1
[INFO] Atoms: H=35, O=30, C=30
[INFO] Atoms: H=35, O=34, C=30
[INFO] Client disconnected.
[INFO] New UDS STREAM client connected.
[INFO] Atoms: H=40, O=34, C=30
[INFO] Atoms: H=40, O=38, C=30
[INFO] Client disconnected.
```

כעת נריץ את ה-molecule_requester:

נריץ ראשית לקוח UDP רגיל ואז נריץ לקוח UDS מסוג datagram:

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules$ cd Uds
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./molecule_requester 127.0.0.1 6666
Using UDP to 127.0.0.1:6666
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): DELIVER WATER 1
[Server Reply] DELIVERED
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): DELIVER GLUCOSE 1
[Server Reply] DELIVERED
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): EXIT
Exiting.
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$
```

```
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules$ cd Uds
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./molecule_requester -f /tmp/uds_dgram.sock
Using UNIX Domain Socket at /tmp/uds_dgram.sock
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): DELIVER WATER 1
[Server Reply] DELIVERED
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): DELIVER GLUCOSE 1
[Server Reply] DELIVERED
Enter command (DELIVER <MOLECULE> <AMOUNT> | EXIT): EXIT
Exiting.
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$
```

וזהו התנהגות השרת← הכל מתקבל, ולאחר וסיימנו, בלי פעילות למשך 60 שניות , אז השרת שלנו יבצע כיבוי אוטומטי:

```
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -o atom_supplier atom_supplier.cpp -lgcov
roy3177@LAPTOP-QCUJB7RP:~/Counting molecules/Uds$ ./uds_server -T 5555 -U 6666 \
-s /tmp/uds_stream.sock \
-d /tmp/uds_dgram.sock \
-h 30 -o 30 -c 30 -t 60
[INFO] UDS STREAM socket bound to /tmp/uds_stream.sock
[INFO] UDS DATAGRAM socket bound to /tmp/uds_dgram.sock
Server is listening on TCP port 5555 and UDP port 6666...
[INFO] New client connected: 127.0.0.1
[INFO] Atoms: H=35, O=30, C=30
[INFO] Atoms: H=35, O=34, C=30
[INFO] Client disconnected.
[INFO] New UDS STREAM client connected.
[INFO] Atoms: H=40, O=34, C=30
[INFO] Atoms: H=40, O=38, C=30
[INFO] Client disconnected.
[INFO] UDP request received: DELIVER WATER
[INFO] Atoms: H=38, O=37, C=30
[INFO] UDP request received: DELIVER GLUCOSE
[INFO] Atoms: H=26, O=31, C=24
[INFO] UDS DATAGRAM request received: DELIVER WATER 1
[INFO] Atoms: H=24, O=30, C=24
[INFO] UDS DATAGRAM request received: DELIVER GLUCOSE 1
[INFO] Atoms: H=12, O=24, C=18
[INFO] Timeout reached. Shutting down.
[INFO] Server shut down.
```