**Написання програми-сервера**

Протокол роботи клієнтської та серверної частини

*Сервер:*

1) Створюємо сокет

2) Прив'язуємо сокет до певної IP-адреси та порту.

3) Встановлюємо сокет в "слухає" режим.

4) Чекаємо на запит на з'єднання.

5) Обробляємо вхідний запит (якщо він надійшов у п. 4), а саме приймаємо від клієнта одне повідомлення та виводимо його на екран.

6) Закриваємо з'єднання з цим клієнтом

7) Якщо максимальний ліміт вхідних з'єднань для даної програми не досягнуто (встановлюється при виклику listen), то йдемо в пункт 4 (і чекаємо на наступне з'єднання), інакше завершуємо роботу.

Сервер приймає з'єднання до тих пір, поки не буде досягнуто ліміт, встановлений при виклику listen, або ви не закрили програму-сервер. При відкритті чергового з'єднання сервер отримує повідомлення від клієнта, виводить його на екран і закриває з'єднання.

*Клієнт:*

1) Створюємо сокет

2) Виконуємо з'єднання із сервером

3) Надсилаємо повідомлення

4) Закриваємо з'єднання із сервером

5) Завершуємо роботу.

При роботі з сокетом більшість програмістів використовують стандартний набір подій: для надсилання даних, прийому даних, з'єднання з іншим сокетом, для встановлення каналу передачі даних при вхідному запиті та для закриття сокету. Можливо, є й інші події для сокетів, але в цій статті ми зосередимося на основних. Коли одне з цих подій, асоційованих із сокетом, відбувається, ми отримуємо сигнал і виробляємо необхідні дії обробки цієї події.

Ось основні константи, що описують мережеві події:

FD\_ACCEPT

FD\_READ

FD\_WRITE

FD\_CLOSE

FD\_CONNECT

Отже, розглянемо весь процес створення, відстеження та обробки мережевих подій.

По-перше, нам доведеться ініціалізувати бібліотеку winsock2.

Ініціалізувати можна будь-де програми, але обов'язково до виклику будь-яких функцій winsock. Наступний важливий момент - це створення події, яку хочемо відслідковувати цьому сокеті. Для цього використовуватимемо виклик Winsock2 API ::WSACreateEvent(); Після того, як подія створена, її потрібно пов'язати із сокетом, події якого ми хочемо контролювати та обробляти. Це робиться функцією WSAEventSelect(...). Далі нам потрібно відстежувати події, що сигналізують про те, що у сокет прийшов запит встановлення каналу зв'язку. Зазвичай таку операцію можна робити на сокеті, що прослуховує (listening).

*Обробка повідомлень про події*

Тепер, коли події задані, нам необхідно чекати на них і, відповідно, обробляти. Для очікування подій можна використовувати функцію WSAWaitForMultipleEvents(...). Ця функція буде працювати як потік у сплячому режимі, доки не відбудеться подія, на яку ми хотіли б відреагувати.

Перший параметр – це кількість подій, які ми хочемо очікувати. Другий параметр – це покажчик на масив подій, на які ми хочемо очікувати. Третій параметр має значення BOOL, яке визначає - чи буде функція залишатися в сплячому режимі, доки не спрацюють всі події. Зазвичай цей параметр задається як false, але можливо, Вам може знадобитися очікувати настання всіх подій. Четвертий параметр визначає - як довго чекати настання події. Зазвичай я запускаю окремий потік та лишаю його як infinite. Але, якщо Ви запускатимете функцію в основному потоці, то може знадобитися поставити обмеження в 5 (або більше) секунд, щоб дати можливість застосуванню обробляти інші події. П'ятий параметр вказує на те, хочемо ми чи ні отримувати алерти.

Далі необхідно подбати про обробників кожної події. Насамперед нам необхідно отримати достовірну інформацію про те, яка подія виникла. І тому існує функція ::WSAEnumNetworkEvents(...).

*Надсилання, читання на обробка повідомлень*

Відправлення повідомлень було реалізоване через string. Тому для відправки повідомлення спочатку ми створюємо змінну, що буде зберігати величину повідомлення та надсилаємо її на сервер за допомогою функції send(), для того, щоб сигналізувати серверу про розмір повідомлення, яке він отримає. Після цього ми надсилаємо саме повідомлення.

Отримане повідомлення ми оброблятимемо з точністю до навпаки. Спочатку програма отримує повідомлення про розмір повідомлення, що має надійти та записує його в ініціалізовану перед цим змінну. Далі створюється змінна типу char\*, що збегігає в собі масив new char розміру [(розмір повідомлення, що має надійти) + 1]. Далі сигналізуємо про кінець повідомлення за допомогою '\0', отримуємо саме повідомлення. Для клієнта після цього та виводимо його в термінал та видаляємо змінну, в якій зберігалося повідомлення, а для сервера ми оброблюємо повідомлення, приводмо його до правильного формату та надсилаємо клієнту(ам), залежно від формату повідомлення(публічне, приватне, бажання вийти) і піся цього видаляємо змінну, в якій зберігалося повідомлення.

*Обробка помилок.*

Всього у коді обох програм описано два варіанти виключень, вони спільні як для клієнта, так і для сервера. Перше виключення це class bibLoadException : public exception, що обробляє помилку завантаження бібліотеки для роботи з програмою, а друге - class cientConnException : public exception – помилка при підключенні клієнта до серверу.

**Опис створення клієнт/серверу на Сі++**

Перед початком роботи необхідно підключити бібліотеку winsock2.h для створення та роботи з сокетами а також iostream для роботи з функціями мови С++, бібліотеку string для роботи з рядками, та підключити using namespace std.

*Сервер.*

Створюється структура allUsers, та її елемент usersLst[100] – список, який зберігає інформацію про підключення клієнтів, а саме його порядковий номер та юзернейм. Connections[100] відповідає за кожного клієнта, що підключився, а Counter зберігає інформацію про фактичну кількість підключень.

SOCKET Connections[100];

int Counter = 0;

allUsers usersLst[100];

Функція, що фактично відповідає за обробку повідомлень та відправлення їх потрібному користувачу - ClientHandler(int index). Її атрибутом є індекс користувача, що надіслав повідомлення. У циклі ми отримуємо повідомлення та його розмір та перетворюємо його у string для зручності обробки. Повідомлення, що має надіслати сервер користувачам має мати відповідний формат, а саме @(username): (message). Якщо текст повідомлення EXIT, то сервер від’єднує клієнта, присвоює його нікнейму «0» та надсилає іншим користувачам повідомення про вихід цього користувача з чату. Якщо перший елемент повідомлення '\_', то ми заходимо в перший цикл – це індикатор приватного повідомлення. Далі ми відокремлюємо нікнейм користувача, якому це повідомлення має прийти на шукаємо його в списку користувачів usersLst[100]. Якщо такий користувач є у списку, то ми надсилаємо йому повідомлення, якщо такого користувача намає, то сервер надсилає користувачу, що відправив це повідомлення, повідомлення про помилку. Якщо ж це повідомлення не було приватним, то ми надсилаємо його у форматі, описаному раніше, усім користувачам, окрім того, хто його написав. Після того видаляємо повідомлення за допомогою delete[] для того, щоб не заповнювати пам'ять.

В головній програмі якщо нема виключних ситуацій то створюється сокет та визначаються усі необхідні поля для роботи з ним.

Сервер надсилає клієнту, що під’єднався, в циклі всіх користувачів із usersLst[100], що в даний момент онлайн. Якщо клієнт введе нікнейм з цього списку, його не буде підключено. Після успішного під’єднання клієнта в циклі йому присвоюється індекс. Клієнт надсилає свій юзернейм серверу, він його додає з індексом в usersLst[100]. Коли до серверу під’єднується новий улієнт, сервер надсилає усім іншим клієнтам повідомлення про його підключення сервер надсилає іншим користувачам повідомлення про те, що новий клієнт доєднався до сервера.

*Клієнт.*

У клієнта також присутня функція ClientHandler(), хоча функціонал вона має набагато менший ніж функція з цією назвою у сервері. В клієнті ця функція буде приймати інформацію, яка їй буде надходити від сервера за допомогою створеного зв’язку через сокети та виводити її в термінал.

В головній програмі початок аналогічний до серверної програми, а саме створення сокету за умови відсутності виключних ситуацій. На термінал користувача виводиться повідомлення, що описує сутніть роботи чату, а саме:

1. Для входу до чат-кімнати потрібно ввести нікнейм у форматі @(username)
2. Якщо буде введений не унікальний нікнейм, то користувач не буде підключений до сервера(усі зайняті нікнейми клієнт отримує при підключенні).
3. Щоб написати приватне повідомлення, потрібно почати його з «тега», що виглядає наступним чином: \_@(username)
4. Якщо користувач хоче вийти, йому потрібно ввести EXIT

Далі створюється булева змінна firstCon, якій присвоюється значення true. Вона сигналізує про те, що це підключення буде першим. У циклі користувач вводить свій нікнейм. Якщо він буде неправильно введений, то користувач буде вводити його знову.

Однак далі у msg1 записується з терміналу повідомлення користувача до серверу, адресоване іншому користувачу, воно проходить перевірку сервером на правильність звертання і тоді вже сервер пересилає це повідомлення потібному користувачу.

**Підсумок.**

Програма дозволяє створювати з’єднання між багатьма користувачами за допомогою створення клієнтно-серверного середовища, в якому сервер отримує від користувачів повідомлення, обробляє їх та відправляє їм. Програма дозволяє декілька режимів роботи чату, а саме: повідомлення можуть бути приватними та публічними, в залежності від формату їх оформлення з терміналу користувача.