Лекція 7
UDP-сокети

Характеристики UDP

Протокол UDP (User Datagram Protocol, протокол датаграм користувача) – це транспортний протокол у комунікаційному домені Internet, визначений у RFC 768. UDP не потребує встановлення логічного з’єднання між сторонами, забезпечує ненадійне невпорядковане доставлення датаграм (відправлені дані мають межі) з перевіркою на помилки в режимах unicast, multicast, широкомовний (тільки в IPv4), не підтримує out-of-band дані. Зазвичай у комунікації з використанням UDP є активна сторона (клієнт) та є пасивна сторона (сервер), після встановлення TCP-з’єднання ролі сторін у загальному розумінні є умовними.

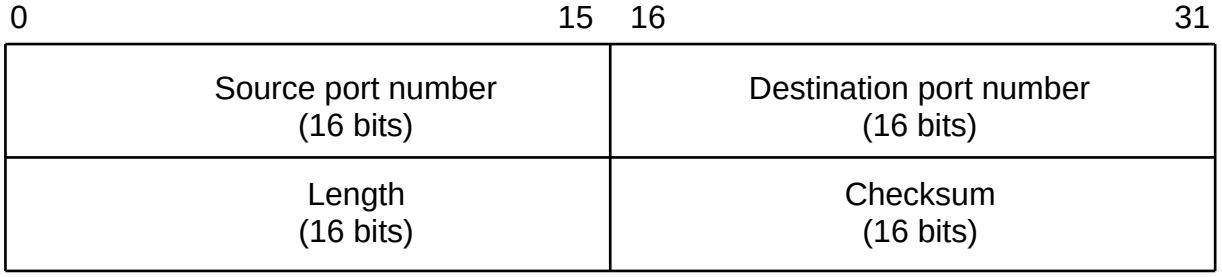
В UDP є *контрольна сума* (*checksum*), яка обчислюється для значень полів псевдозаголовка IP (кілька полів заголовка IPv4 або IPv6), значень полів заголовка UDP та даних (payload) датаграми та зберігається в заголовку UDP. Контрольна сума є опціональною в IPv4 (RFC 1122 рекомендує обчислювати контрольну суму як усталено, заголовок IPv4 має поле зі значенням контрольної суми для значень його полів) та обов’язковою в IPv6 (заголовок IPv6 не має поле зі значенням контрольної суми для значень його полів).

Заголовок UDP та вміст датаграми не шифруються як усталено, тому заголовок UDP та/або вміст датаграми треба шифрувати іншими засобами за потреби. Алгоритм обчислення контрольної суми відомий та використовує тільки дані в IP пакеті, що відправляється, тому отримувач даних, відправлених через UDP, не знає чи ці дані були змінені разом із контрольною сумою сторонніми системами під час спрямування цього пакету в мережі. Також алгоритм обчислення контрольної суми не є надійним порівняно з іншими алгоритмами обчислення контрольних сум, але достатній для практичного використання.

Програма користувача може створити сокет у комунікаційному домені Internet для можливого використання його в UDP, тобто створити UDP-сокет, системним викликом `socket(domain, SOCK_DGRAM, IPPROTO_UDP)`, значення `domain` має бути `AF_INET` для IPv4 або `AF_INET6` для IPv6. Якщо в значенні протоколу вказати нуль, тоді типове значення протоколу для типу сокета `SOCK_DGRAM` буде `IPPROTO_UDP` у комунікаційному домені Internet. У сучасних програмах користувача, краще явно вказувати протокол для комунікаційних доменів, які дозволяють таке робити, для запобігання можливої двозначності.

Заголовок UDP

UDP-датаграма інкапсулюється в дані IP-пакета, відразу після заголовка IPv4 або IPv6. UDP-датаграма складається із заголовка UDP (завжди 8 байт), він однаковий для IPv4 та IPv6, та даних (payload), які відправляє програма.



Якщо значення контрольної суми (Checksum) в заголовку UDP дорівнює нулю, тоді контрольна сума ігнорується. Якщо обчислена контрольна сума дорівнює нулю, тоді в усіх бітах її значення в заголовку UDP встановлюються одиниці (це значення є тим самим, що нуль в алгоритмі обчислення контрольної суми).

Відправлення та отримання даних

Системний виклик `sendto()` відправляє дані через сокет, асоційований з вказаним дескриптором файлу.

```
#include <sys/socket.h>

ssize_t sendto(int sockfd, const void *buf, size_t count, int flags,
        const struct sockaddr *addrbuf, socklen_t addrlen);
```

Аргументи `sockfd`, `buf`, `count` та `flags` такі самі, як у системному виклику `send()`. Якщо сокет застосовується для комунікації, яке потребує встановлення логічного з'єднання, тоді значення аргументів `addrbuf` та `addrlen` ігноруються (рекомендується вказувати null покажчик та `0` для цих аргументів через особливості реалізацій). Якщо сокет застосовується для комунікації, яке не потребує логічного з'єднання, тоді аргумент `addrbuf` має вказувати на об'єкт зі структурою адреси сокета отримувача даних, а аргумент `addrlen` має дорівнювати розміру цього об'єкту.

Системний виклик `recvfrom()` отримує дані із сокета, асоційованого з вказаним дескриптором файлу.

```
#include <sys/socket.h>

ssize_t recvfrom(int sockfd, void *restrict buf, size_t count, int flags,
        struct sockaddr *restrict addrbuf, socklen_t *restrict addrlen);
```

Аргументи `sockfd`, `buf`, `count` та `flags` такі самі, як у системному виклику `recv()`. Якщо аргументи `addrbuf` та `addrlen` null покажчики, тоді вони ігноруються. Інакше аргумент `addrbuf` має вказувати на об’єкт, куди може бути скопійована структура адреси сокета відправника даних. Відповідно аргумент `addrlen` має вказувати на об’єкт, значення якого дорівнюю розміру об’єкта, на який вказує аргумент `addrbuf`. У разі успішного виконання цього системного виклику інформація про адресу відправника зберігається в об’єкт, на який вказує аргумент `addrbuf`, а в об’єкт, на який вказує аргумент `addrlen`, зберігається розмір збереженої інформації. Якщо розмір об’єкта, на який вказує аргумент `addrbuf`, не достатній для збереження інформації про адресу відправника даних, тоді в цей об’єкт зберігається частина інформації про адресу відправника даних. Якщо мережевий протокол сокета не надає адресу відправника даних, тоді вміст об’єкта, на який вказує аргумент `addrbuf`, не специфіковано.

Системний виклик `sendmsg()` відправляє дані та/або допоміжні дані через сокет, асоційований з вказаним дескриптором файлу.

```
#include <sys/socket.h>

ssize_t sendmsg(int sockfd, const struct msghdr *message, int flags);
```

Цей системний виклик є найбільш функціональним з системних викликів `write()`, `writev()`, `send()` та `sendto()`, застосованих для сокета для відправлення даних. Аргументи `sockfd` та `flags` такі самі, як у системному виклику `send()`. Інформація про адресу отримувача (якщо така потрібна), буфери з даними та/або допоміжні дані для відправлення вказуються в об’єкті, на який вказує аргумент `message`.

Системний виклик `recvmsg()` отримує дані та/або допоміжні дані із сокета, асоційованого з вказаним дескриптором файлу.

```
#include <sys/socket.h>

ssize_t recvmsg(int sockfd, struct msghdr *message, int flags);
```

Цей системний виклик є найбільш функціональним з системних викликів `read()`, `readv()`, `recv()` та `recvfrom()`, застосованих для сокета для отримання даних. Аргументи `sockfd` та `flags` такі самі, як у системному виклику `recv()`. Інформація про адресу відправника (якщо така потрібна та якщо така повертається), буфери з отриманими даними та/або допоміжними даними та прапорці результату виконання вказуються в об’єкті, на який вказує аргумент `message`.

У `struct msghdr` є принаймні такі поля: `void *msg_name` та `socklen_t msg_namelen` (покажчик на об’єкт з адресою сокета та розмір цього об’єкта), `struct iovec *msg_iov` та `int msg_iovlen` (інформація про буфери), `void *msg_control` та `socklen_t msg_controllen` (буфер з допоміжними даними та розмір цього буфера), `int msg_flags` (прапорці для отриманих даних). POSIX цього не визначає, але поле `msg_control` має вказувати на буфер з відповідним вирівнюванням (це відповідає тому, як реалізації використовують значення цього поля).

Поля `msg_name` та `msg_namelen` в об’єкті типу `struct msghdr`, залученого в аргументі системних викликів `sendmsg()` та `recvmsg()`, використовуються так само як відповідні аргументи в системних викликах `sendto()` та `recvfrom()` відповідно (значення поля `msg_namelen` у разі успішного виконання `recvmsg()` встановлюється). Поля `msg_iov` та

`msg_iovlen` в об’єкті, залученого в аргументі системних викликів `sendmsg()` та `recvmsg()` використовуються так само як відповідні аргументи в системних викликах `writev()` та `readv()` відповідно.

Значення поля `msg_flags` в об’єкті типу `struct msghdr`, залученого в аргументі системного виклику `sendmsg()` ігнорується. Значення цього поля після успішного виконання системного виклику `recvmsg()` може мати такі прапорці: `MSG_EOR` позначає отримання ознаки кінця запису (має підтримуватися протоколом); `MSG_OOB` позначає отримання out-of-band даних; `MSG_TRUNC` позначає отримання в буфер не всіх байтів даних (має сенс для датаграм або повідомлень); `MSG_CTRUNC` позначає отримання в буфер не всіх байтів допоміжних даних.

Допоміжні дані складаються з масиву пар значень, перше значення – це об’єкт типу `struct cmsghdr`, який визначає тип допоміжних даних, друге значення – це безпосередньо допоміжні дані відповідного типу. У `struct cmsghdr` є принаймні такі поля: `socklen_t cmsg_len` (кількість байтів у поточні парі значень), `int cmsg_level` та `int cmsg_type` (рівень та тип асоційованих допоміжних даних). POSIX цього не визначає, але `cmsg_len` може мати більше значення, ніж потрібне, для врахування

додаткових байтів для отримання правильного вирівнювання для наступного об’єкта типу `struct cmsghdr` у допоміжних даних, якщо такий об’єкт є.

Для роботи з допоміжними даними визначено кілька макросів. Далі ідентифікатор `mhdr` позначає покажчик на об’єкт типу `struct msghdr`, `cmsg` позначає покажчик на об’єкт типу `struct cmsghdr`, який асоційований з об’єктом, на який вказує `mhdr`. Макрос `CMSG_DATA(cmsg)` повертає покажчик типу `unsigned char *` на безпосередні асоційовані допоміжні дані. Макрос `CMSG_NXTHDR(mhdr, cmsg)` повертає покажчик на наступний об’єкт типу `struct cmsghdr` або значення null покажчика, якщо більше об’єктів нема. Макрос `CMSG_FIRSTHDR(mhdr)` повертає покажчик на перший об’єкт типу `struct cmsghdr`, асоційований з об’єктом, на який вказує `mhdr`, або значення null покажчика, якщо допоміжних даних нема. POSIX цього не визначає, але реалізації можуть перевіряти коректність значень полів в об’єкті типу `struct cmsghdr` та в об’єкті типу `struct cmsghdr`, який асоційований з ним, у макросах `CMSG_FIRSTHDR()` та `CMSG_NXTHDR()`, якщо значення полів некоректні, тоді ці макроси повертають значення null покажчиків.

RFC 3542 пропонує ще два макроси для роботи з допоміжними даними, ці макроси не визначені в POSIX, але реалізації зазвичай їх мають. Макрос

`CMSG_SPACE(length)` повертає розмір пам’яті, необхідний для зберігання об’єкта типу `struct cmsghdr` та асоційованих з ним даних розміром `length` байт з врахуванням усіх потрібних вирівнювань. Макрос `CMSG_LEN(length)` повертає значення, яке можна використати для ініціалізації поля `cmsg_len` об’єкта типу `struct cmsghdr` для асоційованих з ним даних розміром `length` байт з врахуванням потрібного вирівнювання для цього об’єкта.