Лекція 2

Мережеві адреси та імена

Бінарне представлення IP адрес

API між програмою користувача та ядром передбачає використання бінарних представлень IP адрес зі заздалегідь визначеними форматами. Текстові представлення IP адрес треба інтерпретувати, що потребує часу, а бінарні представлення IP адрес можна використовувати безпосередньо під час виконання різних завдань, що мають відношення до мережі (створення заголовків мережевих протоколів, мережева маршрутизація, фільтрування трафіку і т. ін.). Також розмір бінарного представлення IP адреси заздалегідь визначений, що спрощує передавання аргументів у системних викликах.

Типи даних для бінарних представлень IP адрес визначені в `<netinet/in.h>`. POSIX вимагає, щоб бінарні представлення IP адрес були збережені в об’єктах цих типів у мережевому порядку байтів. Програма користувача може зберігати бінарні представлення IP адрес в об’єктах цих типів або в об’єктах інших типів у будь-якому порядку байтів для власного використання, але це може призвести до плутанини під час програмування. Є сенс скрізь використовувати мережевий порядок байтів для бінарних представлень IP адрес. Також використання бінарних представлень IP адрес у мережевому порядку байтів

дає змогу працювати з байтами цих представлень за допомогою покажчика на байт, оскільки алгоритми, які використовують арифметику покажчиків під час роботи з бінарними представленнями IP адрес із наперед визначеним порядком байтів, не залежать від порядку байтів у системі.

Бінарне представлення IPv4 адреси вказують в об'єкті типу `struct in_addr`, ця структура має принаймні поле `in_addr_t s_addr`. Тип `in_addr_t` – це беззнаковий цілочисельний тип еквівалентний типу `uint32_t` (IPv4 адреса складається з 32 біт). Бінарне представлення IPv6 адреси вказують в об'єкті типу `struct in6_addr`, ця структура має принаймні поле `uint8_t s6_addr[16]` (IPv6 адреса складається зі 128 біт). Реалізації часто для поля з IP адресою в цих структурах використовують `union`, в якому кожне поле є масив (розмір кожного масиву дорівнює розміру IP адреси), який дає змогу працювати з 8, або 16, або 32 бітовими складовими IP адреси (так званий type punning з `union` у мові програмування C). Це не є стандартним і не має сенс використовувати в переносному програмуванні, але реалізації часто надають таку можливість для оптимізації програм.

Є IP адреси, які визначені у відповідних стандартах, як мережеві адреси для спеціальних використань. Wildcard адреса або unspecified адреса

(неспецифікована адреса) – це мережева адреса, яку використовують для спеціальних цілей на локальній системі. Broadcast адреса (широкомовна адреса) – це мережева адреса, яку використовують для широкомовної мережевої комунікації. Loopback адреса – це адреса локального мережевого інтерфейсу, комунікація через який може бути тільки в локальній системі.

У `<netinet/in.h>` визначені такі макроси: `INADDR_ANY` має значення wildcard IPv4 адреси типу `in_addr_t`; `INADDR_BROADCAST` має значення broadcast IPv4 адреси типу `in_addr_t`; `IN6ADDR_ANY_INIT` має значення типу `struct in6_addr`, яке можна використати для ініціалізації об’єкта в значення wildcard IPv6 адреси; `IN6ADDR_LOOPBACK_INIT` має значення типу `struct in6_addr`, яке можна використати для ініціалізації об’єкта в значення loopback IPv6 адреси. У `<netinet/in.h>` є оголошення зовнішніх константних змінних `in6addr_any` та `in6addr_loopback` типу `struct in6_addr`, які ініціалізовані значеннями `IN6ADDR_ANY_INIT` та `IN6ADDR_LOOPBACK_INIT` відповідно.

POSIX зазначає, що бінарне представлення значень wildcard та broadcast IPv4 адрес не залежить від порядку байтів, це мережеві адреси `0.0.0.0` та `255.255.255.255` відповідно. Wildcard IPv6 адреса – це мережева адреса `::`, loopback IPv6 адреса – це мережева адреса `::1`.

Реалізація може мати нестандартний макрос `INADDR_LOOPBACK`, визначений у `<netinet/in.h>`, зі значенням loopback IPv4 адреси типу `in_addr_t`. Loopback IPv4 адреса – це будь-яка мережева адреса в `127/8`. Зазвичай реалізації для значення макросу `INADDR_LOOPBACK` використовують мережеву адресу `127.0.0.1` у порядку байтів у системі. POSIX та RFC згадують макрос `INADDR_LOOPBACK`, але не визначають його.

Структури адрес сокетів

Структури адрес сокетів (socket address structure) – це структури, які визначені в POSIX (стандартні) та в реалізаціях (нестандартні), об’єкти цих типів мають налаштування для сокетів (лекція 3). Кожний комунікаційний домен має власну структуру адреси сокета. Також є загальна структура адреси сокета, об’єкт цього типу можна використовувати для зберігання вмісту об’єкта будь-якого іншого типу структури адреси сокета. Вміст об’єкта типу структури адреси сокета має ініціалізувати програма, якщо цей вміст буде зчитувати функція POSIX, або цей вміст ініціалізує функція POSIX, якщо вона має відповідну семантику.

Кожна структура адреси сокета має поле, яке визначає номер комунікаційного домену. Це поле має тип `sa_family_t` (беззнаковий цілочисельний тип), визначений у `<sys/socket.h>`. POSIX визначає, що якщо привести (cast) покажчик на будь-яку структуру адреси сокета в покажчик на будь-яку іншу структуру адреси сокета, то покажчики на поля цих структур, які визначають номер комунікаційного домену, повинні збігатися (тобто структури повинні мати однакові вимоги щодо вирівнювання, а ці поля повинні мати однакове зміщення

від початку структури). Реалізації зазвичай розташовують це поле першим, оскільки стандарт C визначає, що перед першим полем структури немає padding bytes. Для кодування номера комунікаційного домену використовують макроси, які починаються із символів `AF_` (наприклад, `AF_INET`, `AF_INET6`, `AF_UNIX`), визначені в `<sys/socket.h>`. Літери `AF` означають address family (сімейство адрес).

Структура адреси сокета комунікаційного домену Internet IPv4 `struct sockaddr_in` визначена в `<netinet/in.h>`. У цій структурі є принаймні такі поля: `sa_family_t sin_family` (номер комунікаційного домену, має бути `AF_INET`), `in_port_t sin_port` (номер порту в мережевому порядку байтів), `struct in_addr sin_addr` (мережева адреса в мережевому порядку байтів).

Структура адреси сокета комунікаційного домену Internet IPv6 `struct sockaddr_in6` визначена в `<netinet/in.h>`. У цій структурі є принаймні такі поля: `sa_family_t sin6_family` (номер комунікаційного домену, має бути `AF_INET6`), `in_port_t sin6_port` (номер порту в мережевому порядку байтів), `uint32_t sin6_flowinfo` (клас трафіку), `struct in6_addr sin6_addr` (мережева адреса в мережевому порядку байтів), `uint32_t sin6_scope_id` (ідентифікація набору інтерфейсів). Поля `sin6_flowinfo` та `sin6_scope_id` будемо встановлювати в нуль (пояснення сенсу цих полів поза тем лекцій).

Тип `in_port_t` – це беззнаковий цілочисельний тип еквівалентний типу `uint16_t` (номер порту складається з 16 біт). Вимога вказувати номер порту в мережевому порядку байтів у структурах адрес сокета комунікаційних доменів Internet IPv4 та IPv6 є історичною, нема технічного обґрунтування цієї вимоги.

POSIX вимагає, щоб усі поля об’єкта типу `struct sockaddr_in6`, вміст якого будуть зчитувати функції API, були ініціалізовані. Якщо програма ініціалізує нестандартні поля об’єкта цього типу нетиповими значеннями, то поведінка будь-якої функції POSIX, яка зчитує вміст цього об’єкта, залежить від реалізації. Будь-яка функція POSIX, яка ініціалізує вміст об’єкта цього типу даних, ініціалізує всі нестандартні поля об’єкта типовими значеннями.

POSIX не вимагає ініціалізувати нестандартні поля об’єкта типу іншої структури адреси сокета (наприклад, об’єкт типу `struct sockaddr_in`), якщо його вміст будуть зчитувати функції POSIX, але рекомендує ініціалізувати їх так само як вимагає ініціалізувати нестандартні поля об’єкта типу `struct sockaddr_in6`.

Ініціалізувати нестандартні поля структури типовими значеннями потрібно так:

```
    struct sockaddr_in sin4 = {};
```

Некоректно ініціалізувати нестандартні поля структури типовими значеннями так:

```
struct sockaddr_in sin4;

memset(&sin4, 0, sizeof(sin4));
```

У нестандартних полях цього об’єкта можуть бути покажчики, а представлення значення null покажчика, не обов’язково збігається зі значенням із нулями в усіх бітах. Також така ініціалізація є некоректною для чисел із рухомою комою.

Загальна структура адреси сокета `struct sockaddr_storage` визначена в `<sys/socket.h>`, розмір цієї структури достатній для того, щоб в об’єкті цього типу можна було зберігати вмісту об’єкта будь-якого іншого типу структури адреси сокета. У цій структурі є принаймні поле `sa_family_t ss_family` (номер комунікаційного домену). Об’єкти цього типу даних зазвичай використовують для узагальнення коду та структур даних для роботи з різними комунікаційними доменами.

Структура адреси сокета `struct sockaddr` визначена в `<sys/socket.h>`. Ця структура адреси сокета не має відношення до конкретного комунікаційного домену. У цій структурі є принаймні такі поля: `sa_family_t sa_family` (номер комунікаційного

домену), `char sa_data[]` (дані змінної довжини). Зазвичай програма користувача використовує тільки покажчики на об’єкти цього типу даних та звертається до значення поля з номером комунікаційного домену. Нема сенсу використовувати об’єкти цього типу в програмі користувача, оскільки розмір цієї структури може бути недостатній для зберігання вмісту об’єкта конкретного типу структури адреси сокета. Відповідно програмі користувача нема сенсу використовувати поле `sa_data`.

У `<netinet/in.h>` визначені макроси, які дають змогу перевірити тип IPv6 адреси, вказаної в об’єкті типу `struct sockaddr_in6`. Ці макроси мають один аргумент, який має мати тип `const struct sockaddr_in6 *`, результати їхнього виконання мають тип `int` (булеве значення). Макрос `IN6_IS_ADDR_UNSPECIFIED()` перевіряє чи вказана IPv6 адреса є wildcard адреса. Макрос `IN6_IS_ADDR_LOOPBACK()` перевіряє чи вказана IPv6 адреса є loopback адреса. Пояснення сенсу інших схожих стандартних макросів поза тем лекцій.

Трансляція мережевих імен та адрес

Структури адрес сокетів комунікаційних доменів Internet IPv4 та IPv6 містять поля з бінарними представленнями мережевих адрес та портів. Програма може ініціалізувати ці поля сама, це не є складним завданням, якщо програма вже має якийсь варіант цих бінарних представлень.

Якщо в програмі потрібно інтерпретувати текстове представлення мережевої адреси або отримати текстове представлення мережевої адреси, то таке завдання не є складним, але є типовим. POSIX визначає кілька функцій, які дають змогу виконати ці завдання.

Функція `inet_ntop()` конвертує бінарне представлення IPv4 або IPv6 адреси в текстове представлення. Функція `inet_pton()` конвертує текстове представлення IPv4 або IPv6 адреси в бінарне представлення.

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *restrict addr,
        char *restrict buf, socklen_t size);
int inet_pton(int af, const char *restrict buf, void *restrict addr);
```

Літера `n` у назвах функцій значить «numeric» (числовий), літера `p` у назвах функцій значить «presentation» (представлення). Аргумент `af` має бути `AF_INET` для IPv4 або `AF_INET6` для IPv6 адреси. Аргумент `addr` має вказувати на буфер (розміром принаймні 4 або 16 байт відповідно), який містить або де буде збережено IPv4 або IPv6 адресу в мережевому порядку байтів. Аргумент `buf` має вказувати на буфер (розміром принаймні `INET_ADDRSTRLEN` або `INET6_ADDRSTRLEN` відповідно), який містить або де буде збережено текстове представлення IPv4 або IPv6 адреси. Аргумент `size` визначає розмір цього буфера. У разі помилки функція `inet_ntop()` повертає null покажчик, номер помилки буде встановлено в `errno`, інакше функція повертає значення аргументу `buf`. У разі помилки функція `inet_pton()` повертає `-1`, номер помилки буде встановлено в `errno`, інакше якщо функція не може конвертувати текстове представлення мережевої адреси в бінарне представлення, то вона повертає `0`, інакше повертає `1`.

Далі, якщо не вказано інше, функція POSIX у разі помилки повертає `-1`, якщо функція повертає значення типу `int`, або null покажчик, якщо функція повертає значення покажчика, а номер помилки буде встановлено в `errno`, яке визначено в `<errno.h>`. Далі, якщо не вказано інше, функція POSIX, яка повертає значення

типу `int`, у разі успішного виконання повертає `0`. Номери помилок – це додатні числа типу `int`, найчастіше це різні числа, визначені в макросах в `<errno.h>`. Є сенс зчитувати значення `errno` тільки тоді, коли функція завершилася з помилкою та номер помилки був встановлений в `errno`. Жодна функція POSIX не встановлює `errno` в нуль. У разі успішного виконання функції POSIX значення `errno` може бути змінено, якщо не вказано інше. Кожний потік багатопотокової програми має власний `errno`.

Приклад програми на C, яка конвертує текстове представлення IPv4 адреси в бінарне представлення, потім конвертує це бінарне представлення в текстове представлення.

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>

int
main(void)
{
    struct in_addr in4_addr;
    const char *addr_str = "1.2.3.4";
    const uint8_t *byte;
    char buf[INET_ADDRSTRLEN];
```

```
    printf("Convert %s to binary presentation\n", addr_str);
    switch (inet_pton(AF_INET, addr_str, &in4_addr.s_addr)) {
    case 1:
        byte = (uint8_t *)&in4_addr.s_addr;
        printf("Binary representation %02x %02x %02x %02x\n",
            byte[3], byte[2], byte[1], byte[0]);
        if (inet_ntop(AF_INET, &in4_addr.s_addr, buf, sizeof(buf)) == nullptr)
            exit_err("inet_ntop()");
        printf("Converted to text presentation %s\n", buf);
        break;
    case 0:
        warn_errx("cannot convert %s", addr_str);
        break;
    default:
        exit_err("inet_pton()");
    }
}
```

Тут у разі помилки програма викликає функцію `exit_err()`, яка виводить вказане повідомлення про помилку та рядок тексту, який відповідає значенню `errno`, та завершує виконання процесу зі статусом помилки. Зрозуміло, що не всі програми мають завершувати своє виконання, якщо не вдалось щось виконати. Реалізацію цієї та подібних функцій можна записати в окремий вихідний файл і потім його використовувати у вихідних кодах різних програм.

Наведемо варіанти реалізації цієї функції мовами програмування C та C++. Цю функцію та інші схожі функції можуть викликати різні потоки та різні процеси, які виводять повідомлення в одному терміналі. Реалізуємо синхронізацію виводу для потоків багатопотокової програми та «синхронізуємо» вивід для кількох процесів на одному термінал у цій функції (це не буде працювати в загальному випадку, тому що POSIX не визначає атомарність запису в пристрій терміналу).

Ця функція отримує аргументи як функція `printf()`. Якщо компілятор надає можливість перевіряти коректність аргументів для функції типу `printf()`, то таку перевірку буде виконано під час компіляції. Також ця функція для C++ виводить текст для поточного винятку (exception) замість рядка тексту для значення `errno`.

Це об’явлення функції `exit_err()` для C та C++.

```
#ifdef __GNUC__
# define PRINTF_LIKE(x, y) __attribute__((format(printf, (x), (y))))
#else
# define PRINTF_LIKE(x, y)
#endif

[[noreturn]] void exit_err(const char *msg, ...) PRINTF_LIKE(1, 2);
```

Ця частина коду спільна для C та C++.

```
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MSG_DEFAULT "output_msg(): cannot format message\n"

#ifdef PTHREADS

#include <locale.h>
#include <pthread.h>

#define OUTPUT_LOCK() pthread_mutex_lock(&output_mtx)
#define OUTPUT_UNLOCK() pthread_mutex_unlock(&output_mtx)
#define STRERROR(e) strerror_l((e), uselocale((locale_t)0))

static pthread_mutex_t output_mtx = PTHREAD_MUTEX_INITIALIZER;

#else

#define OUTPUT_LOCK()
#define OUTPUT_UNLOCK()
#define STRERROR(e) strerror(e)
```

```
#endif
```

Реалізація функції `exit_err()` на C.

```
static void
output_msg(int fd, const char *prefix, bool errflag, int errnum,
    const char *format, va_list args)
{
    char buf[256];
    size_t len;

    snprintf(buf, sizeof(buf), "%s", prefix);
    len = strlen(buf);
    vsnprintf(buf + len, sizeof(buf) - len, format, args);
    len += strlen(buf + len);
    if (errflag) {
        snprintf(buf + len, sizeof(buf) - len, ": %s", STRERROR(errnum));
        len += strlen(buf + len);
    }
    buf[len] = '\n';
    OUTPUT_LOCK();
    if (len < sizeof(buf) - 1)
        write(fd, buf, len + 1);
    else
        write(fd, MSG_DEFAULT, sizeof(MSG_DEFAULT) - 1);
    OUTPUT_UNLOCK();
}
```

```
[[noreturn]] void
exit_err(const char *format, ...)
{
    int errnum = errno;
    va_list args;

    va_start(args, format);
    output_msg(STDERR_FILENO, "Error: ", true, errnum, format, args);
    va_end(args);
    exit(EXIT_FAILURE);
}
```

Реалізація функції `exit_err()` на C++.

```
#include <exception>

void
output_msg(int fd, const char* prefix, bool errflag, int errnum,
    std::exception_ptr exptr, const char* format, va_list args)
{
    char buf[256];
    size_t len;

    snprintf(buf, sizeof(buf), "%s", prefix);
    len = strlen(buf);
    vsnprintf(buf + len, sizeof(buf) - len, format, args);
    len += strlen(buf + len);
```

```
    if (errflag) {
        if (exptr) {
            try {
                std::rethrow_exception(exptr);
            } catch (std::exception& ex) {
                snprintf(buf + len, sizeof(buf) - len, ": %s", ex.what());
            } catch (const char* str) {
                snprintf(buf + len, sizeof(buf) - len, ": %s", str);
            } catch (...) {
                snprintf(buf + len, sizeof(buf) - len, ": <exception>");
            }
        } else {
            snprintf(buf + len, sizeof(buf) - len, ": %s", STRERROR(errnum));
        }
        len += strlen(buf + len);
    }
    buf[len] = '\n';
    OUTPUT_LOCK();
    if (len < sizeof(buf) - 1)
        write(fd, buf, len + 1);
    else
        write(fd, MSG_DEFAULT, sizeof(MSG_DEFAULT) - 1);
    OUTPUT_UNLOCK();
}

[[noreturn]] void
exit_err(const char* format, ...)
{
```

```
    int errnum{errno};
    std::exception_ptr exptr{std::current_exception()};
    va_list args;

    va_start(args, format);
    output_msg(STDERR_FILENO, "Error: ", true, errnum, exptr, format, args);
    va_end(args);
    exit(EXIT_FAILURE);
}
```

Також у програмах у лекціях використані такі функції: `exit_errx()`, яка схожа на `exit_err()`, але не виводить рядок тексту для значення `errno` або поточного винятку; `warn_err()` та `warn_errx()`, які схожі на `exit_err()` та `exit_errx()` відповідно, але не завершують виконання процесу; `exit_errn()` та `warn_errn()`, які схожі на `exit_err()` та `warn_err()`, але мають номер помилки в першому аргументі. Вихідний код функції `exit_err()` може бути використаний для реалізації цих функцій.

Якщо в програмі потрібно інтерпретувати текстове представлення номера порту, то це можна виконати функцією `strtoul()`.

Замість текстового представлення номера порту можна вказувати ім’я сервісу. IANA (Internet Assigned Numbers Authority) призначає імена, транспортні

протоколи та номери портів деяким відомим або стандартним мережевим сервісам, призначає імена та номери мережевим протоколам. Ім’я сервісу зв’язано з номером порту та транспортним протоколом, одне ім’я сервісу може бути асоційовано з кількома парами номера порту та транспортного протоколу. Інформація про сервіси зазвичай вказана у файлі **/etc/services**. Інформація про мережеві протоколи зазвичай вказана у файлі **/etc/protocols**.

Функція `getservbyname()` повертає інформацію про сервіс за ім’ям сервісу та вказаним транспортним протоколом. Функція `getservbynumber()` повертає інформацію про сервіс за номером порту та вказаним транспортним протоколом.

```
#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
```

Ці функції повертають покажчик на об’єкт з інформацією про сервіс. Процес не має модифікувати дані в цьому об’єкті та дані, на які вказують покажчики в полях цього об’єкта, наступний виклик функції `getserv*()` може зробити значення цих даних та покажчиків недійсними. Значення аргументу `port` має бути

конвертовано зі значення типу `uint16_t` в мережевому порядку байтів. Якщо аргумент `proto` null покажчик, то буде повернуто інформацію для будь-якого транспортного протоколу. Ці функції повертають значення null покажчика, якщо шукана інформація не була знайдена або в разі помилки, але значення `errno` не позначатиме помилку. Ці функції не мають бути thread-safe.

У `struct servent` є принаймні такі поля: `char *s_name` (покажчик на рядок з офіційним іменем сервісу), `char **s_aliases` (покажчик на масив покажчиків на рядки з альтернативними іменами сервісу, масив завершується значенням null покажчика), `int s_port` (значення, яке в разі конвертування в значення типу `uint16_t` є номер порту сервісу в мережевому порядку байтів), `char *s_proto` (покажчик на рядок з ім’ям транспортного протоколу).

Список функцій, які не мають бути thread-safe, зазначено в SUSv4 у розділі *System Interfaces* / *General Information* / *Threads* / *Thread-Safety*. Якщо в лекції для якоїсь функції POSIX не вказано, що вона не має бути thread-safe, то вона є thread-safe. Програма може вільно викликати thread-safe функцію та інші будь-які thread-safe функції в своїх потоках.

Приклад програми на C, яка виводить номер порту та альтернативні імена сервісу для мережевого сервісу «http» та транспортного протоколу «tcp».

```c
#include <arpa/inet.h>
#include <inttypes.h>
#include <netdb.h>
#include <stdint.h>
#include <stdio.h>

int
main(void)
{
    const char *servname = "http";
    const char *protoname = "tcp";
    const struct servent *servent;

    servent = getservbyname(servname, protoname);
    if (servent == nullptr)
        exit_errx("could not find %s / %s service or error occurred",
            servname, protoname);
    printf("Service %s / %s, port number %" PRIu16,
        servname, protoname, ntohs((uint16_t)servent->s_port));
    if (servent->s_aliases != nullptr && servent->s_aliases[0] != nullptr) {
        printf(", aliases");
        for (size_t i = 0; servent->s_aliases[i] != nullptr; ++i)
            printf(" %s", servent->s_aliases[i]);
    }
```

```
    printf("\n");
}
```

Приклад виконання цієї програми:

```
$ ./a.out
Service http / tcp, port number 80, aliases www www-http
```

У цій програмі перевіряти значення поля `s_aliases` непотрібно відповідно до POSIX, але це не помилка.

Функція `getservent()` дає змогу отримати інформацію про всі сервіси з бази даних.

```
#include <netdb.h>

void endservent(void);
struct servent *getservent(void);
void setservent(int stayopen);
```

Функція `setservent()` встановлює позицію в базі даних на перший елемент. Якщо аргумент `stayopen` не нуль, то база даних буде відкритою після виклику будь-якої функції `getserv*()` (це оптимізація, тобто буде відкритий принаймні один дескриптор файлу для доступу до бази даних). Функція `getservent()` повертає покажчик на об’єкт з інформацією про сервіс у поточній позиції бази даних та

змінює позицію в базі даних на наступний елемент, або null покажчик у разі завершення інформації в базі даних або в разі помилки, але значення `errno` не позначатиме помилку. Функція `endservent()` закриває базу даних. Ці функції не мають бути thread-safe.

Функція `getprotobyname()` повертає інформацію про протокол за ім'ям протоколу. Функція `getprotobynumber()` повертає інформацію про протокол за номером протоколу.

```
#include <netdb.h>

struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
```

Ці функції мають таку саму семантику, як відповідні функції для отримання інформації про сервіси, тільки для протоколів.

У `struct protoent` є принаймні такі поля: `char *p_name` (покажчик на рядок з офіційним ім'ям протоколу), `char **p_aliases` (покажчик на масив покажчиків на рядки з альтернативними іменами протоколу, масив завершується значенням null покажчика), `int p_proto` (номер протоколу).

Функція `getprotoent()` дає змогу отримати інформацію про всі протоколи з бази даних.

```
#include <netdb.h>

void endprotoent(void);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

Ці функції мають таку саму семантику, як відповідні функції для отримання інформації про сервіси, тільки для протоколів.

Якщо програмі потрібно отримати мережеві адреси для мережевого імені або отримати мережеве ім’я для мережевої адреси, то таке завдання не є простим, його рішення залежить від того, як у системі влаштована робота з мережевими іменами. POSIX визначає відповідні функції, які дають змогу виконати ці завдання. Також ці функції дають змогу виконати супутні завдання.

Функція `getaddrinfo()` транслює мережеве ім’я або текстове представлення мережевої адреси та/або ім’я сервісу або текстове представлення номера порту в структури адрес сокетів та в супутню інформацію.

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

int getaddrinfo(const char *restrict nodename, const char *restrict servname,
        const struct addrinfo *restrict ai_hints,
        struct addrinfo **restrict ai_result);
```

Один з аргументів `nodename`, який вказує на рядок із мережевим ім’ям або текстовим представленням мережевої адреси, та `servname`, який вказує на рядок з ім’ям сервісу або текстовим представленням номера порту, має бути не null покажчиком. Функція `getaddrinfo()` повертає нуль у разі успішного виконання, інакше повертає номер помилки, визначений в `<netdb.h>` (макроси `EAI_*`). Помилка `EAI_SYSTEM` позначає системну помилку, номер системної помилки буде встановлено в `errno`.

У `struct addrinfo` є принаймні такі поля: `int ai_flags` (прапорці `AI_*`), `int ai_family` (номер комунікаційного домену), `int ai_socktype` (тип сокета), `int ai_protocol` (номер транспортного протоколу), `socklen_t ai_addrlen` (цілочисельний тип із принаймні 32 значущими бітами, розмір структури адреси сокета), `struct sockaddr *ai_addr` (якщо це значення привести до відповідного типу покажчика, то це буде покажчик на об’єкт зі структурою адреси сокета), `char *ai_canonname` (покажчик на рядок із

канонічним мережевим ім’ям), `struct addrinfo *ai_next` (покажчик на наступну структуру в списку або значення null покажчика).

Якщо поле `ai_hints` не null покажчик, то в об’єкті, на який воно вказує, вказані опції для `getaddrinfo()`. POSIX вимагає, щоб усі поля в цій структурі були ініціалізовані. Якщо поля `ai_addrlen`, `ai_addr`, `ai_canonname`, `ai_next` або якесь нестандартне поле ініціалізовані не типовими значеннями, то поведінка `getaddrinfo()` залежить від реалізації. Значення `AF_UNSPEC` у полі `ai_family` значить повертати результати для будь-яких комунікаційних доменів. Нульове значення в полі `ai_socktype` значить повертати результати для будь-яких типів сокетів. Нульове значення в полі `ai_protocol` значить повертати результати для будь-яких транспортних протоколів. Інші значення в полях `ai_family`, `ai_socktype` та `ai_protocol` фільтрують результат. Для будь-яких значень цих полів та вмісту рядків, на які вказують `nodename` та `servname`, результат має коректні комбінації значень комунікаційних доменів, типів сокета та номерів транспортних протоколів. Якщо `ai_hints` null покажчик, то `getaddrinfo()` поводиться так, ніби структура має значення `AF_UNSPEC` у полі `ai_family` та нульові значення в полях `ai_flags`, `ai_socktype` та `ai_protocol`. Поле `ai_flags` використовують тільки в об’єкті, на який вказує `ai_hints`. Значення

цього поля в об’єктах, які будуть повернуті в списку, на який вказує `*ai_result`, не визначено.

Прапорець `AI_PASSIVE` вказує, що структури адрес сокетів, яку будуть повернуті, можуть бути використані для призначення сокету для пасивного відкриття мережевого з’єднання. Якщо `nodename` null покажчик і прапорець `AI_PASSIVE` встановлено, будуть повернені структури адрес сокетів для wildcard адрес. Якщо `nodename` null покажчик і прапорець `AI_PASSIVE` не встановлено, будуть повернені структури адреси сокета для loopback адрес. Якщо `nodename` не null покажчик, то прапорець `AI_PASSIVE` буде ігноровано.

Прапорець `AI_CANONNAME` вказує повернути покажчик на рядок із канонічним мережевим ім’ям (наприклад, FQDN). Якщо `nodename` не null покажчик і прапорець `AI_CANONNAME` встановлено, то в першому об’єкті зі списку, на який вказує `*ai_result`, поле `ai_canonname` вказуватиме на рядок із канонічним мережевим ім’ям, якщо його вдалось визначити, інакше на рядок, на який вказує `nodename`, або на рядок із таким самим вмістом.

Прапорець `AI_NUMERICHOST` вказує інтерпретувати рядок, на який вказує `nodename`, як числове представлення мережевої адреси. Прапорець `AI_NUMERICSERV` вказує інтерпретувати рядок, на який вказує `servname`, як числове представлення номера порту.

Якщо прапорець `AI_V4MAPPED` вказаний разом зі значенням `AF_INET6` в полі `ai_family` в об’єкті, на який вказує аргумент `ai_hint`, то будуть повернуті структури адрес сокетів з IPv4 адресами відображеними в IPv6 адреси, якщо не вдалось знайти IPv6 адрес. Якщо прапорець `AI_ALL` вказаний разом із прапорцем `AI_V4MAPPED`, то будуть повернуті структури адрес сокетів з IPv4 адресами, відображеними в IPv6 адреси разом зі знайденими IPv6 адресами.

Прапорець `AI_ADDRCONFIG` вказує повертати результати тільки для комунікаційних доменів, якщо в системі є мережеві інтерфейси, які сконфігуровані на використання мережевих адрес у цих комунікаційних доменах (POSIX дає визначення цього поля для комунікаційних доменів Internet IPv4 та IPv6, але цей прапорець працюватиме для будь-яких комунікаційних доменів).

Використання нечислового представлення в рядку, на який вказує `nodename` та/або `servname`, з невстановленими прапорцями `AI_NUMERICHOST`, `AI_NUMERICSERV`, або використання прапорця `AI_CANONNAME` може потребувати звернення до деякого мережевого сервісу (наприклад, до серверів NIS та/або DNS). Таке звернення потребує часу, тому програма має це враховувати та, можливо, викликати функцію `getaddrinfo()` в окремому потоці.

У разі успішного виконання `getaddrinfo()` зберігає результат у списку об’єктів, покажчик на перший об’єкт списку буде збережено в `*ai_result`. Значення полів `ai_family`, `ai_socktype`, `ai_protocol`, `ai_addrlen` та вміст структури адреси сокета, на яку вказує поле `ai_addr`, можна використовувати безпосередньо у відповідних функціях POSIX. Будь-які поля в структурах адрес сокетів, які не були встановлені явно в аргументах (наприклад, поле `sin6_flowinfo` в `struct sockaddr_in6`), будуть мати нульові значення.

Функція `getaddrinfo()` з відповідними значеннями аргументів реалізує те саме, що реалізують функції `inet_pton()`, `getservbyname()` та `getservbyport()`.

Функція `freeaddrinfo()` звільняє пам’ять, яку адресують об’єкти зі списку, який повернула функція `getaddrinfo()` у разі успішного виконання.

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

void freeaddrinfo(struct addrinfo *ai);
```

Функція `gai_strerror()` повертає покажчик на рядок з описом помилки, яку кодують макросом `EAI_*`, визначеним у `<netdb.h>`.

```
#include <netdb.h>

const char *gai_strerror(int error);
```

Приклад програми на C, яка викликає функцію `getaddrinfo()`. Програма отримує мережеве ім’я або мережеву адресу та ім’я сервісу або номер порту в опціях командного рядка та транслює їх в об’єкти типу структура адреси сокета для комунікаційного домену Internet IPv4. Також програма має прапорець командного рядка для повернення структур адрес сокетів, які можна призначати сокету для пасивного відкриття мережевого з’єднання.

```
#include <arpa/inet.h>
```

```c
#include <sys/socket.h>
#include <errno.h>
#include <inttypes.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct addrinfo ai_hints = {};
    struct addrinfo *ai_result;
    const char *hostname = nullptr;
    const char *servname = nullptr;
    const struct addrinfo *ai;
    const struct sockaddr_in *sin4;
    int opt, error;
    char buf[INET_ADDRSTRLEN];

    opterr = 0;
    while ((opt = getopt(argc, argv, "h:p:P")) != -1) {
        switch (opt) {
        case 'h':
            hostname = optarg;
            break;
        case 'p':
            servname = optarg;
            break;
```

```
        case 'P':
            ai_hints.ai_flags = AI_PASSIVE;
            break;
        default:
            exit_errx("wrong option -%c", optopt);
        }
    }
    if (optind < argc)
        exit_errx("wrong number of command-line arguments");

    ai_hints.ai_family = AF_INET;
    error = getaddrinfo(hostname, servname, &ai_hints, &ai_result);
    if (error != 0) {
        if (errno == EAI_SYSTEM)
            exit_err("getaddrinfo(): %s", gai_strerror(error));
        exit_errx("getaddrinfo(): %s", gai_strerror(error));
    }

    for (ai = ai_result; ai != nullptr; ai = ai->ai_next) {
        sin4 = (struct sockaddr_in *)ai->ai_addr;
        if (inet_ntop(AF_INET, &sin4->sin_addr.s_addr, buf, sizeof(buf)) ==
            nullptr
        ) {
            exit_err("inet_ntop()");
        }
        printf("address family %d, socket type %d, protocol %d, "
            "address %s, port %" PRIu16 "\n", ai->ai_family, ai->ai_socktype,
            ai->ai_protocol, buf, ntohs(sin4->sin_port));
```

```
    }

    freeaddrinfo(ai_result);
}
```

Виклик функції `freeaddrinfo()` в цій програмі не потрібен, ця функція викликана для демонстрації її використання.

Приклад використання цієї програми:

```
$ ./a.out -h 1.2.3.4 -p 1234
address family 2, socket type 1, protocol 6, address 1.2.3.4, port 1234
address family 2, socket type 2, protocol 17, address 1.2.3.4, port 1234
address family 2, socket type 3, protocol 0, address 1.2.3.4, port 1234
$ ./a.out -h 1.2.3.4
address family 2, socket type 1, protocol 6, address 1.2.3.4, port 0
address family 2, socket type 2, protocol 17, address 1.2.3.4, port 0
address family 2, socket type 3, protocol 0, address 1.2.3.4, port 0
$ ./a.out -P -p http
address family 2, socket type 1, protocol 6, address 0.0.0.0, port 80
address family 2, socket type 2, protocol 17, address 0.0.0.0, port 80
address family 2, socket type 1, protocol 132, address 0.0.0.0, port 80
address family 2, socket type 5, protocol 132, address 0.0.0.0, port 80
$ ./a.out -P -p 80
address family 2, socket type 1, protocol 6, address 0.0.0.0, port 80
address family 2, socket type 2, protocol 17, address 0.0.0.0, port 80
```

```
address family 2, socket type 3, protocol 0, address 0.0.0.0, port 80
$ ./a.out -P -p 0
address family 2, socket type 1, protocol 6, address 0.0.0.0, port 0
address family 2, socket type 2, protocol 17, address 0.0.0.0, port 0
address family 2, socket type 3, protocol 0, address 0.0.0.0, port 0
$ ./a.out
Error: getaddrinfo(): Name or service not known
$ ./a.out -h some.some
Error: getaddrinfo(): Name or service not known
$ ./a.out -h some
Error: getaddrinfo(): Temporary failure in name resolution
```

Функція `getnameinfo()` транслює вміст об’єкта типу структура адреси сокета в мережеве ім’я або текстове представлення мережевої адреси та/або в ім’я сервісу або текстове представлення номера порту.

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

int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
        char *restrict nodename, socklen_t nodelen, char *restrict servname,
        socklen_t servlen, int flags);
```

Функція `getnameinfo()` повертає нуль у разі успішного виконанні, інакше повертає номер помилки, так само як номер помилки повертає функція `getaddrinfo()`.

Аргумент `sa` має вказувати на об’єкт відповідного типу структури адреси сокета (тип покажчика на відповідну структуру адреси сокета треба привести в тип покажчика `struct sockaddr *`). Аргумент `salen` має дорівнюватися розміру цього об’єкта.

Аргумент `flags` вказує опції трансляції: `NI_NOFQDN` вказує повертати коротку частину мережевого імені в локальній мережі; `NI_NUMERICHOST` вказує повертати текстове представлення мережевої адреси; `NI_NAMEREQD` вказує повертати помилку, якщо не вдалось визначити мережеве ім’я; `NI_NUMERICSERV` вказує повертати текстове представлення номера порту; `NI_NUMERICSCOPE` вказує повертати номер інтерфейсу, а не його ім’я (цей прапорець буде ігноровано для не IPv6 адрес); `NI_DGRAM` вказує, що сервіс, який представлений номером порту, використовує протокол датаграм (має сенс під час трансляції номера порту в ім’я сервісу, іноді той самий номер порту може відповідати різним сервісам, залежно від транспортного протоколу).

Якщо `nodename` не null покажчик та `nodelen` не нуль, то `nodename` має вказувати на буфер розміром принаймні `nodelen` байт, куди `getnameinfo()` збереже результат транслювання мережевої адреси. Якщо `servname` не null покажчик та `servlen` не

нуль, то `servname` має вказувати на буфер розміром принаймні `servlen` байт, куди `getnameinfo()` збереже результат транслювання номера порту. Пам’яті для збереження результату трансляції має бути достатньо.

Невикористання прапорців `NI_NUMERICHOST` та `NI_NUMERICSERV` може потребувати звернення до деякого мережевого сервісу (наприклад, до серверів NIS та/або DNS). Таке звернення потребує часу, тому програма має це враховувати та, можливо, викликати функцію `getnameinfo()` в окремому потоці.

Функція `getnameinfo()` з відповідними значеннями аргументів реалізує те саме, що реалізують функції `inet_ntop()` та `getservbyport()`.

Ім’я системи

POSIX не визначає функцій або аргументів до функцій для отримання інформації про мережеві інтерфейси системи. Зазвичай інформацію про мережеві інтерфейси можна отримати використовуючи системний виклик `ioctl()` з відповідними аргументами. POSIX визначає дві функції для отримання імені системи, яке може мати відношення до комунікаційного домену. Одному з мережевих інтерфейсів системи може бути призначена мережева адреса в цьому комунікаційному домені.

Функція `gethostname()` повертає ім’я системи.

```
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

Аргумент `name` вказує на буфер, куди буде скопійоване ім’я системи. Якщо значення `namelen` менше ніж довжина імені системи, враховуючи кінцевий символ NUL, то в буфер буде скопійовано частину імені системи, але не специфіковано чи буде скопійовано кінцевий символ NUL.

Функція `uname()` повертає різну інформацію про систему.

```c
#include <sys/utsname.h>

int uname(struct utsname *utsname);
```

Інформацію про систему буде скопійована в буфер, на який вказує аргумент `utsname`. У `struct utsname` є принаймні такі поля: `char sysname[]` (назва ОС), `char nodename[]` (ім'я системи), `char release[]` (інформація про реліз ОС), `char version[]` (інформація про версію релізу), `char machine[]` (назва архітектури комп'ютера). Розміри полів цієї структури не специфіковано, але дані в цих полях завершені кінцевим символом NUL. Формати даних у кожному полі цієї структури залежать від реалізації.

Функція `uname()` у разі успішного виконання повертає невід'ємне значення, повертає `-1` у разі помилки.

POSIX визначає, щоб `uname()` може бути визначено також як макрос.

Для того, щоб коректно використовувати деякі функції POSIX, необхідно знати про можливі обмеження. POSIX визначає функції для отримання інформації

про обмеження в системі, де виконується програма (обмеження для програми під час її виконання), макроси з типовими значеннями обмежень у системі, для якої була скомпільована програма (обмеження в системі, відомі під час компілювання програми) та макроси з гарантованими мінімальними значеннями обмежень для POSIX-сумісної системи.

Функція `sysconf()` повертає інформацію про системні обмеження під час виконання програми для вказаного аргументу, який позначає системне обмеження.

```
#include <unistd.h>

long sysconf(int name);
```

Якщо обмеження для вказаного аргументу не визначено (нема обмеження), `sysconf()` повертає `-1` та не змінює `errno` (тобто треба встановлювати `errno` в нуль перед викликом цієї функції, а потім перевірити значення `errno`, якщо функція повернула `-1`). Невизначене обмеження не передбачає нескінченне обмеження (нема обмеження). Якщо `sysconf()` не підтримує вказаний аргумент, то вона завершується з помилкою, номер помилки `EINVAL`. Якщо аргумент має

відношення до функціоналу, який система не підтримує, то значення, яке повертає `sysconf()`, не специфіковано.

Значення обмеження, яке повертає функція `sysconf()`, не буде змінено впродовж виконання процесу, винятком є результат виклику `sysconf()` з аргументом `_SC_OPEN_MAX` (лекція 3). Значення, яке повертає `sysconf()`, не може бути більш обмежувальним, ніж значення відповідного макросу, визначено в `<limits.h>` для системи. А значення цього макросу не має бути більш обмежувальним, ніж відповідне значення, яке визначено в POSIX, воно також визначено в `<limits.h>` (деякі реалізації не відповідають цій умові).

Деякі системи мають некоректну реалізацію функції `sysconf()`. Інформацію про деякі обмеження можна отримати тільки використовуючи відповідні системні виклики, натомість реалізації можуть повертати наперед запрограмовану інформацію про обмеження.

Функція `sysconf()` з аргументом `_SC_HOST_NAME_MAX` повертає значення на одиницю більше, ніж максимальна можлива довжина імені системи. Значення, яке повертає `sysconf()` для цього аргументу, не може бути більш обмежувальним, ніж

константне значення макросу `HOST_NAME_MAX`, яке може бути визначено в `<limits.h>`. для системи (це типове значення відповідного обмеження в системі, для якої була скомпільована програма). Якщо макрос `HOST_NAME_MAX` не визначено, то відповідне обмеження не визначено. Мінімальне допустиме константне значення визначено в макросі `_POSIX_HOST_NAME_MAX`, який визначений у `<limits.h>` (це значення визначено в POSIX).

Приклад використання функції `sysconf()` для визначення максимально можливої довжини імені системи на C.

```c
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <unistd.h>

int
main(void)
{
    long val;

#ifdef _SC_HOST_NAME_MAX
    errno = 0;
    val = sysconf(_SC_HOST_NAME_MAX);
    if (val < 0) {
```

```
        if (errno != 0) {
            if (errno == EINVAL)
                warn_errx("_SC_HOST_NAME_MAX is not supported");
            else
                exit_err("sysconf() with _SC_HOST_NAME_MAX");
        } else {
            printf("_SC_HOST_NAME_MAX has no limit (indeterminate)\n");
        }
    } else {
        printf("_SC_HOST_NAME_MAX %ld\n", val);
    }
#else
    printf("_SC_HOST_NAME_MAX is not defined\n");
#endif
#ifdef HOST_NAME_MAX
    printf("HOST_NAME_MAX %ld\n", (long)HOST_NAME_MAX);
#else
    printf("HOST_NAME_MAX is not defined (indeterminate)\n");
#endif
    printf("_POSIX_HOST_NAME_MAX %ld\n", (long)_POSIX_HOST_NAME_MAX);
}
```