6. Черги повідомлень в Linux

6.1. Повідомлення як засоби зв'язку і засоби синхронізації процесів

У матеріалах попередніх тем були розглянуті такі засоби організації взаємодії процесів із засобів System V IPC, як розділювана пам'ять (тема 4) і семафори (тема 5). Третім і останнім, найбільш семантично навантаженим засобом, що входить в System V IPC, є черги повідомлень. Можна розглядати моделі повідомлень як спосіб взаємодії процесів через лінії зв'язку, у якому на передану інформацію накладається певна структура, тому процес, що приймає дані, може чітко визначити, де закінчується одна порція інформації і починається інша. Така модель дозволяє задіяти ту саму лінію зв'язку для передачі даних у двох напрямках між декількома процесами. Ми також розглядали можливість використання повідомлень із вбудованими механізмами взаємовиключення і блокування при читанні з порожнього буфера і запису в переповнений буфер для організації синхронізації процесів.

У матеріалах цієї теми розглянемо використання черг повідомлень System V IPC для забезпечення обох вказаних функцій.

6.2. Черги повідомлень в Linux як складова частина System V IPC

Оскільки черги повідомлень входять до засобів System V IPC, для них вірно все, що говорилося раніше про ці засоби в цілому і вже знайоме нам. Черги повідомлень, як і семафори, і розділювана пам'ять, є засобом зв'язку з непрямою адресацією, вимагають ініціалізації для організації взаємодії процесів і спеціальних дій для звільнення системних ресурсів після закінчення взаємодії. Простором імен черг повідомлень є та ж сама множина значень ключа, що генеруються за допомогою функції **`ftok()`**. Для виконання примітивів **`send`** і **`receive`**, відповідним системним викликам як параметр передаються IPC-дескриптори (див. тему 4, "Дескриптори System V IPC") черг повідомлень, що однозначно ідентифікують їх у всій обчислювальній системі.

Черги повідомлень розташовуються в адресному просторі ядра операційної системи у вигляді однонаправлених списків і мають обмеження за обсягом інформації, що зберігається в кожній черзі. Кожний елемент списку є окремим повідомленням. Повідомлення мають атрибут, який називається типом повідомлення. Вибірка повідомлень із черги (виконання примітива **`receive`**) може здійснюватися трьома способами:

1. У порядку **`FIFO`**, незалежно від типу повідомлення.
2. У порядку **`FIFO`** для повідомлень конкретного типу.
3. Першим вибирається повідомлення з мінімальним типом, що не перевищує деякого заданого значення, яке прийшло раніше інших повідомлень із тим же типом.

Реалізація примітивів **`send`** і **`receive`** забезпечує приховане від

користувача взаємовиключення під час розміщення повідомлення в чергу або його одержання із черги. Також вона забезпечує блокування процесу при спробі виконати примітив **receive** над порожньою чергою або чергою, у якій відсутнє повідомлення потрібного типу, або при спробі виконати примітив **send** для черги, у якій немає вільного місця.

Черги повідомлень, як і інші засоби System V IPC, дозволяють організувати взаємодію процесів, що не перебувають одночасно в обчислювальній системі.

6.3. Створення черги повідомлень або доступ до вже існуючої. Системний виклик msgget()

Для створення черги повідомлень, асоційованої з певним ключем, або доступу за ключем до вже існуючої черги використовується системний виклик **msgget()**, що є аналогом системних викликів **shmget()** для розділюваної пам'яті і **semget()** для масиву семафорів. Він повертає значення IPC-дескриптора для цієї черги. При цьому існують ті ж способи створення і доступу, що й для розділюваної пам'яті або семафорів.

Системний виклик msgget()

Прототип системного виклику

```
#include <types.h>
#include <ipc.h>
#include <msg.h>
int msgget(key_t key, int msgflg);
```

Опис системного виклику

Системний виклик **msgget** призначений для виконання операції доступу до черги повідомлень і, у випадку її успішного завершення, повертає дескриптор System V IPC для цієї черги (ціле невід'ємне число, що однозначно характеризує чергу повідомлень всередині обчислювальної системи і використовується надалі для інших операцій з нею).

Параметр **key** є ключем System V IPC для черги повідомлень, тобто фактично її іменем із простору імен System V IPC. Як значення цього параметра може бути використане значення ключа, отримане за допомогою функції **ftok()**, або спеціальне значення **IPC_PRIVATE**. Використання значення **IPC_PRIVATE** завжди приводить до спроби створення нової черги повідомлень із ключем, що не збігається із значенням ключа ні однієї із уже існуючих черг і не може бути отриманий за допомогою функції **ftok()** ні при одній комбінації її параметрів.

Параметр **msgflg** – прапорці – відіграє роль тільки при створенні нової черги повідомлень і визначає права різних користувачів при доступі до черги, а також необхідність створення нової черги і поведінки системного виклику при спробі створення. Він є деякою комбінацією (за допомогою операції побітове або - "|") визначених значень і вісімкових

прав доступу аналогічно, як і для **semget()** та **shmget()**.

Черга повідомлень має обмеження на загальну кількість збереженої інформації, яка може бути змінена адміністратором системи. Поточне значення обмеження можна довідатися за допомогою команди

ipcs -l

Значення, що повертається

Системний виклик повертає значення дескриптора System V IPC для черги повідомлень при нормальному завершенні і значення -1 при виникненні помилки.

6.4. Реалізація примітивів send і receive. Системні виклики msgsnd() і msgrcv()

Для виконання примітива **send** використовується системний виклик **msgsnd()**, що копіює користувацьке повідомлення в чергу повідомлень, задану IPC-дескриптором. При вивченні опису цього виклику зверніть особливу увагу на наступні моменти:

- Тип даних **struct msgbuf** не є типом даних для користувацьких повідомлень, а є лише шаблоном для створення таких типів. Користувач сам повинен створити структуру для своїх повідомлень, у якій першим полем повинна бути змінна типу **long**, яка містить додатне значення типу повідомлення.
- Третій параметр – довжина повідомлення – вказується не вся довжина структури даних, що відповідає повідомленню, а тільки довжина корисної інформації, тобто інформації, яка розташовується в структурі даних після типу повідомлення. Це значення може бути рівним 0 у випадку, коли вся корисна інформація полягає в самому факті приходу повідомлення (повідомлення використовується як сигнальний засіб зв'язку).
- У матеріалах практичних занять ми, як правило, будемо використовувати нульове значення прапорця системного виклику, яке приводить до блокування процесу при відсутності вільного місця в черзі повідомлень.

Системний виклик msgsnd()

Прототип системного виклику

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, struct msgbuf *ptr,
           int length, int flag);
```

Опис системного виклику

Системний виклик **msgsnd** призначений для переміщення

повідомлення в чергу повідомлень, тобто є реалізацією примітива **send**.

Параметр **msqid** є дескриптором System V IPC для черги, у яку відправляється повідомлення, тобто значенням, що повернув системний виклик **msgget()** при створенні черги або при її пошуку за ключем.

Структура **struct msgbuf** описана у файлі **<sys/msg.h>**:

```
struct msgbuf {
long mtype;
char mtext[1];
};
```

Вона є деяким шаблоном структури повідомлення користувача. Повідомлення користувача – це структура, перший елемент якої обов'язково має тип **long** і містить тип повідомлення, а далі розміщується інформативна частина теоретично довільної довжини (практично в Linux вона обмежена розміром 4080 байт і може ще бути зменшена системним адміністратором), яка містить саме повідомлення. Наприклад:

```
struct mymsgbuf {
long mtype;
char mtext[1024];
} mybuf;
```

При цьому інформація зовсім не обов'язково є текстовою, наприклад:

```
struct mymsgbuf {
long mtype;
struct {
int iinfo;
float finfo;
} info;
} mybuf;
```

Тип повідомлення повинен обов'язково бути додатним числом. Справжня довжина корисної частини інформації (тобто інформації, розташованої в структурі після типу повідомлення) повинна бути передана системному виклику як параметр **length**. Цей параметр може дорівнювати 0, якщо вся корисна інформація – в самому факті наявності повідомлення. Системний виклик копіює повідомлення, розташоване за адресою, на яку вказує параметр **ptr**, у чергу повідомлень, задану дескриптором **msqid**.

Параметр **flag** може приймати два значення: **0** і **IPC_NOWAIT**. Якщо значення прапорця дорівнює **0** і в черзі не вистачає місця для того, щоб помістити повідомлення, то системний виклик блокується доти, поки не звільниться місце. При значенні прапорця **IPC_NOWAIT** системний виклик у цій ситуації не блокується, а констатує виникнення помилки із встановленням значення змінної **errno**, описаної у файлі **<errno.h>**, рівним **EAGAIN**.

Значення, що повертається

Системний виклик повертає значення 0 при нормальному

завершенні і значення -1 при виникненні помилки.

Примітив **receive** реалізується системним викликом **msgrcv()**. При вивченні опису цього виклику потрібно звернути особливу увагу на наступні моменти:

- Тип даних **struct msgbuf**, як і для виклику **msgsnd()**, є лише шаблоном для користувацького типу даних.
- Спосіб вибору повідомлення задається нульовим, додатним або від'ємним значенням параметра **type**. Точне значення типу обраного повідомлення можна визначити з відповідного поля структури, у яку системний виклик скопіює повідомлення.
- Системний виклик повертає довжину тільки корисної частини скопійованої інформації, тобто інформації, розташованої в структурі після поля типу повідомлення.
- Обране повідомлення вилучається із черги повідомлень.
- Як параметр **length** вказується максимальна довжина корисної частини інформації, що може бути розміщена в структурі, адресованій параметром **ptr**.
- Надалі ми будемо, як правило, користуватися нульовим значенням прапорців для системного виклику, що приводить до блокування процесу у випадку відсутності в черзі повідомлень із потрібним типом і до помилкової ситуації у випадку, коли довжина інформативної частини обраного повідомлення перевищує довжину, вказану в параметрі **length**.

Системний виклик msgrcv()

Прототип системного виклику

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv(int msqid, struct msgbuf *ptr,
           int length, long type, int flag);
```

Опис системного виклику

Системний виклик **msgrcv** призначений для одержання повідомлення із черги повідомлень, тобто є реалізацією примітива **receive**.

Спосіб вибірки:

- У порядку **FIFO**, незалежно від типу повідомлення.
- У порядку **FIFO** для повідомлень із типом **n**.
- Першим вибирається повідомлення з мінімальним типом, що не перевищує значення **n**, яке прийшло раніше всіх інших повідомлень із тим же типом.

Параметр **msqid** є дескриптором System V IPC для черги, з якої

повинне бути отримане повідомлення, тобто значення, яке повернув системний виклик **msgget()** при створенні черги або при її пошуку за ключем.

Параметр **type** визначає спосіб вибірки повідомлення із черги.

Параметр **length** повинен містити максимальну довжину корисної частини інформації (тобто інформації, розташованої в структурі після типу повідомлення), яка може бути розміщена в повідомленні.

У випадку успіху системний виклик копіює обране повідомлення із черги повідомлень за адресою, зазначеною в параметрі **ptr**, одночасно видаляючи його із черги повідомлень.

Параметр **flag** може приймати значення **0** або бути якою-небудь комбінацією прапорців **IPC_NOWAIT** і **MSG_NOERROR**. Якщо прапорець **IPC_NOWAIT** не встановлений і черга повідомлень порожня або в ній немає повідомлень із потрібним типом, то системний виклик блокується до появи потрібного повідомлення. При встановленні прапорця **IPC_NOWAIT** системний виклик у цій ситуації не блокується, а констатує виникнення помилки із встановленням значення змінної **errno**, описаної у файлі **<errno.h>**, рівним **EAGAIN**. Якщо реальна довжина корисної частини інформації в обраному повідомленні перевищує значення, зазначене в параметрі **length** і прапорець **MSG_NOERROR** не встановлений, то вибірка повідомлення не здійснюється, і фіксується наявність помилкової ситуації. Якщо прапорець **MSG_NOERROR** встановлений, то в цьому випадку помилки не виникає, а повідомлення копіюється в скороченому виді.

Значення, що повертається

Системний виклик повертає при нормальному завершенні реальну довжину корисної частини інформації (тобто інформації, розташованої в структурі після типу повідомлення), скопійованої із черги повідомлень, і значення -1 при виникненні помилки.

Максимально можлива довжина інформативної частини повідомлення в операційній системі Linux становить 4080 байт і може бути зменшена при генерації системи. Поточне значення максимальної довжини можна визначити за допомогою команди

```
ipcs -l
```

6.5. Вилучення черги повідомлень із системи за допомогою команди ipcrm або системного виклику msgctl()

Після завершення процесів, що використали чергу повідомлень, вона не вилучається із системи автоматично, а продовжує зберігатися в системі разом з усіма повідомленнями, до яких не було запитів доти, поки не буде виконана спеціальна команда або спеціальний системний виклик. Для вилучення черги повідомлень можна скористатися вже знайомою нам командою **ipcrm**, що у цьому випадку набуде вигляду:

ipcrm msg <IPC ідентифікатор>

Для одержання IPC ідентифікатора черги повідомлень застосуйте команду **ipcs**. Можна вилучити чергу повідомлень і за допомогою системного виклику **msgctl()**. Цей виклик уміє виконувати і інші операції над чергою повідомлень, але в рамках даного курсу ми їх розглядати не будемо. Якщо деякий процес перебував у стані очікування при виконанні системного виклику **msgrcv()** або **msgsnd()** для черги, що вилучається, то він буде розблокований, і системний виклик повідомить про наявність помилкової ситуації.

Системний виклик msgctl()

Прототип системного виклику

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Опис системного виклику

Системний виклик **msgctl** призначений для одержання інформації про чергу повідомлень, зміни її атрибутів і вилучення із системи.

Ми будемо користуватися системним викликом **msgctl** тільки для вилучення черги повідомлень із системи. Параметр **msqid** є дескриптором System V IPC для черги повідомлень, тобто значенням, що повернув системний виклик **msgget()** при створенні черги або при її пошуку за ключем.

Як параметр **cmd** у рамках курсу будемо передавати значення **IPC_RMID** – команду для вилучення черги повідомлень із заданим ідентифікатором. Параметр **buf** для цієї команди не використовується, тому завжди будемо підставляти значення NULL.

Значення, що повертається

Системний виклик повертає значення 0 при нормальному завершенні і значення -1 при виникненні помилки.

6.6. Приклад з однонапрямленою передачею текстової інформації

Розглянемо дві прості програми.

```
/* Ця програма одержує доступ до черги повідомлень, відправляє в неї 5 текстових повідомлень із типом 1 і одне порожнє повідомлення з типом 255, яке буде служити для програми 06-1b.c сигналом припинення роботи. */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
```

```
#include <stdio.h>
#define LAST_MESSAGE 255 /* Тип повідомлення для
    припинення роботи програми 06-1b.c */
int main()
{
    int msqid; /* IPC дескриптор для черги повідомлень */
char pathname[] = "06-1a.c"; /* Ім'я файлу,
       що використовується для генерації ключа. Файл із
       таким іменем повинен існувати в поточній
       директорії */
    key_t key; /* IPC ключ */
    int i,len; /* Лічильник циклу і довжина
        інформативної частини повідомлення */
    struct mymsgbuf
    {
        long mtype;       // Користувацька структура для
        char mtext[81];  // повідомлення
    } mybuf;
    /* Генеруємо IPC ключ із імені файлу 06-1a.c у
    поточній директорії і номера екземпляра черги
    повідомлень 0. */
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Намагаємося одержати доступ за ключем до черги
    повідомлень, якщо вона існує, або створити її, із
    правами доступу read & write для всіх користувачів*/
    if((msqid = msgget(key, 0666 | IPC_CREAT)) < 0){
        printf("Can\'t get msqid\n");
        exit(-1);
    }
    /* Посилаємо в циклі 5 повідомлень із типом 1
    у чергу повідомлень, яка ідентифікується msqid.*/
    for (i = 1; i <= 5; i++){
        /* Спочатку заповнюємо структуру для нашого
        повідомлення і визначаємо довжину інформативної
        частини */
        mybuf.mtype = 1;
        strcpy(mybuf.mtext, "This is text message");
        len = strlen(mybuf.mtext)+1;
        /* Відсилаємо повідомлення. У випадку помилки
        повідомляємо про це і видаляємо чергу
        повідомлень із системи. */
        if (msgsnd(msqid, (struct msgbuf *) &mybuf,
            len, 0) < 0){
            printf("Can\'t send message to queue\n");
            msgctl(msqid, IPC_RMID,
```

```
                (struct msqid_ds *) NULL);
            exit(-1);
        }
    }
    /* Відсилаємо повідомлення, що змусить процес, який
    одержує повідомлення, припинити роботу, з типом
    LAST_MESSAGE і довжиною 0 */
    mybuf.mtype = LAST_MESSAGE;
    len = 0;
    if (msgsnd(msqid, (struct msgbuf *) &mybuf,
        len, 0) < 0){
        printf("Can\'t send message to queue\n");
        msgctl(msqid, IPC_RMID,
            (struct msqid_ds *) NULL);
        exit(-1);
    }
    return 0;
}
```

Лістинг 6.1а. Програма 06-1a.c для ілюстрації роботи із чергами повідомлень.

```
/* Ця програма одержує доступ до черги повідомлень і читає
з неї повідомлення з будь-яким типом у порядку FIFO доти,
поки не одержить повідомлення з типом 255, яке буде
сигналом припинення роботи. */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <stdio.h>
#define LAST_MESSAGE 255
int main()
{
    int msqid;
    char pathname[] = "06-1a.c";
    key_t key;
    int len, maxlen; /* Реальна довжина і максимальна
        довжина інформативної частини повідомлення */
    struct mymsgbuf
    {
        long mtype;
        char mtext[81];
    } mybuf;
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
```

```
    if((msqid = msgget(key, 0666 | IPC_CREAT)) < 0){
        printf("Can\'t get msqid\n");
        exit(-1);
    }
    while(1){
        /* У нескінченному циклі приймаємо повідомлення
будь-якого типу в порядку FIFO з максимальною довжиною
інформативної частини 81 символ доти, поки не надійде
повідомлення з типом LAST_MESSAGE*/
        maxlen = 81;
        if(( len = msgrcv(msqid,
            (struct msgbuf *) &mybuf, maxlen, 0, 0) < 0){
            printf("Can\'t receive message from
                    queue\n");
            exit(-1);
        }
        /* Якщо прийняте повідомлення має тип
        LAST_MESSAGE, припиняємо роботу і видаляємо
        чергу повідомлень із системи. У протилежному
        випадку друкуємо текст прийнятого повідомлення*/
        if (mybuf.mtype == LAST_MESSAGE){
            msgctl(msqid, IPC_RMID,
            (struct msqid_ds *) NULL);
        exit(0);
        }
        printf("message type = %ld, info = %s\n",
        mybuf.mtype, mybuf.mtext);
    }
    return 0; /* Тільки для відсутності
        warning при компіляції. */
}
```

Лістинг 6.1b. Програма 06-1b.c для ілюстрації роботи із чергами повідомлень.

Перша із цих програм посилає п'ять текстових повідомлень із типом 1 і одне повідомлення нульової довжини з типом 255 другій програмі. Друга програма в циклі приймає повідомлення будь-якого типу в порядку **FIFO** і друкує їх вміст доти, поки не одержить повідомлення з типом 255. Повідомлення з типом 255 є для неї сигналом до завершення роботи і ліквідації черги повідомлень. Якщо перед запуском кожної із програм черга повідомлень ще була відсутня в системі, то програма створить її.

Зверніть увагу на використання повідомлення з типом 255 як сигналу для припинення роботи другого процесу. Це повідомлення має нульову довжину, тому що його інформативність вичерпується самим фактом наявності повідомлення.

6.7. Передача числової інформації

В описі системних викликів `msgsnd()` і `msgrcv()` говориться про те, що передана інформація не обов'язково повинна бути текстом.

Ми можемо скористатися чергами повідомлень для передачі даних будь-якого виду. При передачі різнорідної інформації доцільно інформативну частину поєднувати всередині повідомлення в окрему структуру для правильного обчислення довжини інформативної частини:

```
struct mymsgbuf {
    long mtype;
    struct {
        short sinfo;
        float finfo;
    } info;
} mybuf;
```

У деяких обчислювальних системах числові дані розміщуються в пам'яті з вирівнюванням на певні адреси (наприклад, на адреси, кратні 4). Тому реальний розмір пам'яті, необхідної для розміщення декількох числових даних, може виявитися більшим суми довжин цих даних, тобто в нашому випадку

```
sizeof(info)>=sizeof(short)+sizeof(float)
```

Для повної передачі інформативної частини повідомлення як довжину потрібно вказувати не суму довжин полів, а повну довжину структури.

6.8. Двосторонній зв'язок через одну чергу повідомлень

Наявність у повідомлень типів дозволяє організувати двосторонній зв'язок між процесами через ту саму чергу повідомлень. Процес 1 може посилати процесу 2 повідомлення з типом 1, а одержувати від нього повідомлення з типом 2. При цьому для вибірки повідомлень в обох процесах варто користуватися другим способом вибору (див. "Черги повідомлень в Linux як складова частина System V IPC").

6.9. Поняття мультиплексування. Мультиплексування повідомлень. Модель взаємодії процесів клієнт-сервер. Нерівноправність клієнта і сервера

Використовуючи алгоритм з попереднього прикладу, ми можемо організувати одержання повідомлень одним процесом від множини інших процесів через одну чергу повідомлень і відправлення їм відповідей через ту ж чергу повідомлень, тобто здійснити мультиплексування повідомлень. Взагалі під мультиплексуванням інформації розуміють можливість одночасного обміну інформацією з декількома партнерами. Метод мультиплексування широко застосовується в моделі взаємодії процесів клієнт-сервер. У цій моделі один із процесів є сервером. Сервер одержує запити від інших процесів – клієнтів – на виконання деяких дій і відправляє їм результати обробки запитів. Найчастіше модель клієнт-сервер використовується при розробці мережних додатків, з які ми розглянемо в

матеріалах завершальних практичних занять курсу. Вона початково припускає, що взаємодіючі процеси нерівноправні:

- Сервер, як правило, працює постійно, напротязі життя додатка, а клієнти можуть працювати епізодично.
- Сервер чекає запиту від клієнтів, ініціатором же взаємодії є клієнт.
- Як правило, клієнт звертається до одного сервера за раз. До сервера можуть одночасно надходити запити від декількох клієнтів.
- Клієнт повинен знати, як звернутися до сервера (наприклад, якого типу повідомлення він приймає) перед початком організації запиту до сервера. Сервер може одержати відсутню інформацію про клієнта із запиту, що прийшов.

Розглянемо наступну схему мультиплексування повідомлень через одну чергу повідомлень для моделі клієнт-сервер. Нехай сервер одержує із черги повідомлень тільки повідомлення з типом 1. До повідомлень із типом 1, що посилають серверу, процеси-клієнти включають значення своїх ідентифікаторів процесу. Прийнявши повідомлення з типом 1, сервер аналізує його зміст, виявляє ідентифікатор процесу, який послав запит, і відповідає клієнтові, посилаючи повідомлення з типом рівним ідентифікатору процесу, що відправляв запит. Процес-клієнт після відправлення запиту очікує відповіді у вигляді повідомлення з типом, рівним своєму ідентифікатору. Оскільки ідентифікатори процесів у системі різні і жоден користувацький процес не може мати PID рівний 1, всі повідомлення можуть бути прочитані тільки тими процесами, яким вони адресовані. Якщо обробка запитів забирає тривалий час, сервер може організовувати паралельну обробку запитів, породжуючи для кожного запиту новий процес-нащадок або нову нитку виконання.

Завдання для самостійної роботи

Самостійно опрацюйте матеріали теми 6 до початку практичного заняття, а також такі питання

1. Монітори.
2. Повідомлення.
3. Еквівалентність семафорів, моніторів та повідомлень.

Завдання для лабораторної роботи

1. Що таке черги повідомлень? Для чого вони використовуються?
2. Що складає простір імен черг повідомлень?
3. Як відбувається створення черги повідомлень або доступ до уже існуючої?
4. Реалізація примітивів **`send`** і **`receive`**. Для чого призначені системні виклики **`msgsnd()`** і **`msgrcv()`**?
5. Як відбувається вилучення черги повідомлень із системи за допомогою команди або системного виклику?

6. Наберіть програми з іменами **`06-1a.c`** і **`06-1b.c`** для ілюстрації роботи із чергами повідомлень, відкомпілюйте і перевірте правильність їхньої поведінки.
7. Модифікуйте попередні програми **`06-1a.c`** і **`06-1b.c`** для передачі нетекстових повідомлень.
8. Напишіть, відкомпілюйте і виконайте програми, які здійснюють двосторонній зв'язок через одну чергу повідомлень.
9. Поясніть поняття мультиплексування. Як здійснюється мультиплексування повідомлень? Поясніть модель взаємодії процесів клієнт-сервер. У чому полягає нерівноправність клієнта і сервера?
10. Напишіть, відкомпілюйте і виконайте програми сервера і клієнтів для запропонованої схеми мультиплексування повідомлень.