Обмін даними в ОС UNIX за допомогою черг повідомлень.

1. Загальні відомості

1.1. На попередніх контрольних роботах ми познайомилися з механізмами, що забезпечують потокову передачу даних між процесами в операційній системі `UNIX`, а саме з `pip`'ами і `FIFO`. Потокові механізми досить прості в реалізації і зручні для використання, але мають ряд істотних недоліків:

- – Операції читання і запису не аналізують вміст переданих даних. Процес, що прочитав 20 байт із потоку, не може сказати, чи були вони записані одним процесом чи декількома, чи записувалися вони за один раз чи було, наприклад, виконано 4 операцій запису по 5 байт.
- – Для передачі інформації від одного процесу до іншого потрібно, як мінімум, дві операції копіювання даних: перший раз - з адресного простору передавального процесу в системний буфер, друг раз - із системного буфера в адресний простір приймаючого процесу.
- – Процеси, що обмінюються інформацією, повинні одночасно існувати в обчислювальній системі. Не можна записати інформацію в потік за допомогою одного процесу, завершити його, а потім, через якийсь час, запустити інший процес і прочитати записану інформацію.

1.2. Зазначені вище недоліки потоків даних привели до розробки інших механізмів передачі інформації між процесами. Частина цих механізмів, що вперше з'явилися в UNIX System V і згодом перекочували практично в усі сучасні версії операційної системи UNIX, одержала загальну назву System V IPC (IPC - interprocess communications). У групу System V IPC входять: черги повідомлень, пам'ять що розділяється і семафори. Ці засоби організації взаємодії процесів зв'язані не тільки спільністю походження, але і мають схожий інтерфейс для виконання подібних операцій, наприклад, для виділення і звільнення відповідного ресурсу в системі.

1.3. Усі засоби зв'язку з System V IPC, як і вже розглянуті нами pipe і FIFO, є засобами зв'язку з непрямою адресацією. Для організації взаємодії неспоріднених процесів за допомогою засобу зв'язку з непрямою адресацією необхідно, щоб цей засіб зв'язку мав ім'я. Відсутність імен у `pip`'ів дозволяє процесам одержувати інформацію про розташування `pip`'а в системі і його стані тільки через “родинні” зв'язки. Наявність асоційованого імені в `FIFO` - імені спеціалізованого файлу у файловій системі - дозволяє неспорідненим процесам одержувати цю інформацію через інтерфейс файлової системи.

1.4. Множину усіх можливих імен для об'єктів якого-небудь виду прийнято називати простором імен відповідного виду об'єктів. Для `FIFO` простором імен є множина усіх припустимих імен файлів у файловій системі. Для всіх об'єктів з System V IPC таким простором імен є множина значень деякого цілочисельного типу даних - `key_t` - ключа. Причому програмісту не дозволено прямо присвоювати значення ключа, це значення задається через комбінацію імені файлу, що вже існує у файловій системі, і невеликого цілого числа - наприклад, номера екземпляра засобу зв'язку.

1.5. Такий хитрий спосіб одержання значення ключа зв'язаний із двома причинами:

- – Якщо дозволити програмістам самим присвоювати значення ключа для ідентифікації засобів зв'язку, то не виключено, що два програмісти випадково скористаються тим самим значенням, не підозрюючи про це. Тоді їхні процеси будуть несанкціоновано взаємодіяти через той самий засіб комунікації, що може привести до нестандартної поведінки цих процесів. Тому основним компонентом значення ключа є

перетворене в числове значення повне ім'я деякого файлу, доступ до якого на читання дозволений процесу. Кожен програміст має можливість використовувати для цієї мети свій специфічний файл. Слід зазначити, що перетворення з текстового імені файлу в число ґрунтується на розташуванні зазначеного файлу на диску чи іншому фізичному носії. Тому для утворення ключа варто застосовувати файли, що не змінюють свого положення протягом часу організації взаємодії процесів.

– – Другий компонент значення ключа використовується для того, щоб дозволити програмісту зв'язати з тим самим ім'ям файлу більше одного екземпляра кожного засобу зв'язку. В якості такого компоненту можна задавати порядковий номер відповідного екземпляра.

Одержання значення ключа з двох компонентів здійснюється функцією `ftok()`.

1.6. Ще раз підкреслимо три важливих моменти, зв'язаних з використанням імені файлу для одержання ключа:

– Необхідно вказувати ім'я файлу, що вже існує у файловій системі і для якого процес має право доступу на читання.

– Зазначений файл повинен зберігати своє положення на диску доти, поки всі процеси, що беруть участь у взаємодії, не одержать ключ System V IPC.

– Вказування імені файлу, як одного з компонентів для одержання ключа, ні в якому разі не означає, що інформація, передана за допомогою асоційованого засобу зв'язку, буде розташовуватися в цьому файлі. Інформація буде зберігатися всередині адресного простору операційної системи, а задане ім'я файлу лише дозволяє різним процесам згенерувати ідентичні ключі.

1.7. З попередніх лабораторних робіт відомо, що інформацію про потоки вводу-виводу, з якими має справу процес, зокрема про `pipe` і `FIFO`, операційна система зберігає в таблиці відкритих файлів процесу. Системні виклики, що здійснюють операції над потоком, використовують як параметр індекс елемента таблиці відкритих файлів, що відповідає потоку, - файловий дескриптор. Використання файлових дескрипторів для ідентифікації потоків всередині процесу дозволяє застосовувати до них вже існуючий інтерфейс для роботи з файлами, але в той же час призводить до автоматичного закриття потоків при завершенні процесу.

1.8. При реалізації компонентів System V IPC була прийнята інша концепція. Ядро операційної системи зберігає інформацію про всі засоби System V IPC, що використовуються у системі, поза контекстом процесів користувача. При створенні нового засобу зв'язку чи одержанні доступу до вже існуючого процес одержує ціле число - дескриптор (ідентифікатор) цього засобу зв'язку, що однозначно ідентифікує його у всій обчислювальній системі. Цей дескриптор повинен передаватися як параметр усім системним викликам, що здійснюють подальші операції над відповідним засобом System V IPC.

1.9. Така концепція дозволяє усунути один із самих істотних недоліків, потокових засобів зв'язку - вимога одночасного існування взаємодіючих процесів, але в той же час вимагає підвищеної обережності для того, щоб процес, що одержує інформацію, не прийняв замість нових старі дані, випадково залишені в механізмі комунікації.

1.10. Черги повідомлень розташовуються в адресному просторі ядра операційної системи у вигляді однонаправлених списків і мають обмеження на обсяг інформації, що зберігається в кожній черзі. Повідомлення мають атрибут - тип повідомлення. Вибірка повідомлень з черги може здійснюватися трьома способами:

– У порядку `FIFO`, незалежно від типу повідомлення.

– У порядку `FIFO` для повідомлень конкретного типу.

– Першим вибирається повідомлення з мінімальним типом, що не перевищує деякого заданого значення, що прийшло раніше інших повідомлень з тим самим типом.

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

1.11. Процеси, що використовують цей тип міжпроцесного зв'язку, можуть виконувати дві операції:

– Послати повідомлення.
– Прийняти повідомлення.

Перед тим, як посилати або приймати повідомлення, потрібно створити чергу повідомлень з унікальним ідентифікатором і асоційованою з ним структурою даних (`msgget`). Породжений унікальний ідентифікатор називається ідентифікатором черги повідомлень (`msqid`); він використовується для звертань до черги повідомлень і асоційованої з нею структурою даних.

1.12. Говорячи про чергу повідомлень варто мати на увазі, що реально в ній зберігаються не самі повідомлення, а їх описи, що мають наступну структуру:

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

struct msg
{
   // Вказівник на наступне повідомлення
   struct msg *msg_next;
   // Тип повідомлення
   long msg_type;
   // Довжина тексту повідомлення
   short msg_ts;
   // Адреса тексту повідомлення
   short msg_spot;
};
```

1.13. З кожним унікальним ідентифікатором черги повідомлень асоційована одна структура даних, що містить наступну інформацію:

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

struct msqid_ds
{
   // Структура прав на виконання
операцій
   struct ipc_perm msg_perm;
   // Покажчик на перше повідомлення в
черзі
   struct msg *msg_first;
   // Покажчик на останнє повідомлення
в черзі
   struct msg *msg_last;
   // Поточне число байт у черзі
   ushort msg_cbytes;
   // Число повідомлень у черзі
   ushort msg_qnum;
   // Макс. припустиме число байт у
черзі
   ushort msg_qbytes;
   // Id останнього відправника
   ushort msg_lspid;
```

```
    // Id останнього одержувача
    ushort msg_lrpid;
    // Час останнього відправлення
    time_t msg_stime;
    // Час останнього одержання
    time_t msg_rtime;
    // Час останньої зміни
    time_t msg_ctime;
};
```

1.14. Поле `msg_perm` даної структури використовує як шаблон структуру `ipc_perm`, що задає права на операції з повідомленнями і визначається так:

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

  struct ipc_perm
  {
     // Ідентифікатор користувача
     ushort uid;
     // Ідентифікатор групи
     ushort gid;
     // Ідентифікатор творця черги
     ushort cuid;
     // Id групи творця черги
     ushort cgid;
     // Права на читання/запис
     ushort mode;
     // Послідовність номерів слотів
     ushort seq;
     // Ключ
     key_t  key;
};
```

1.15. Після того, як створені черга повідомлень з унікальним ідентифікатором і асоційована з нею структура даних, можна використовувати системні виклики сімейства `msgop(2)` (операції над чергами повідомлень) і `msgct(2)` (керування чергами повідомлень).

Операції, як згадувалося вище, полягають у посилці і прийомі повідомлень. Для кожної з цих операцій передбачений системний виклик, `msgsnd()` і `msgrcv()` відповідно.

2. Синтаксис та призначення системних викликів

2.1. Функція `ftok`.

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(pathname, proj)
           char *pathname;
           char proj;
```

Функція `ftok` служить для перетворення імені існуючого файлу і цілого числа (наприклад, порядкового номера екземпляра засобів зв'язку) у ключ System V IPC.

Параметр `pathname` повинен бути покажчиком на ім'я існуючого файлу, доступного для процесу, що викликає функцію.

Параметр `proj` - це невелике ціле число, що характеризує екземпляр засобу зв'язку.

У випадку неможливості генерації ключа функція повертає негативне значення, у протилежному випадку вона повертає значення згенерованого ключа. Тип даних key_t являє собою 32-бітове ціле.

2.2. Системний виклик `msgget`.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key, msgflg)
            key_t key;
            int msgflg;
```

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 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 ім'ям із простору імен System V IPC. Як значення цього параметра може бути використане значення ключа, отримане за допомогою функції `ftok()`, чи спеціальне значення `IPC_PRIVATE`. Використання значення `IPC_PRIVATE` завжди приводить до спроби створення нової черги повідомлень із ключем, що не збігається зі значенням ключа ні однієї з вже існуючих черг і не може бути отриманий за допомогою функції `ftok()` ні при одній комбінації її параметрів.

Параметр `msgflg` - прапори – має значення тільки при створенні нової черги повідомлень і визначає права різних користувачів при доступі до черги, а також необхідність створення нової черги і поведінку системного виклику при спробі створення. Він є деякою комбінацією (за допомогою операції побітове або - "|") наступних визначених значень і прав доступу (див. лабораторну роботу №3):

Тип	Oct	Hex
IPC_CREAT	01000	0x200
IPC_EXCL	02000	0x400

Прапор `IPC_CREAT` вказує, необхідність створення нової черги повідомлень якщо для зазначеного ключа її не існує. `IPC_EXCL` – застосовується разом із прапором `IPC_CREAT`. При спільному їхньому використанні та існуванні черги з зазначеним ключем доступ до черги не дозволяється і констатується помилкова ситуація, при цьому змінна `errno`, прийме значення `EEXIST`.

Черга повідомлень має обмеження по загальній кількості збереженої інформації. Обмеження може бути змінено адміністратором системи. Поточне значення обмеження можна довідатися за допомогою команди `ipcs -l`.

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

2.3. Системний виклик `msgsnd`.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd (msqid, msgp, msgsz, msgflg)
           int msqid;
           struct msgbuf* msgp;
            int msgsz;
            int msgflg;
```

Системний виклик msgsnd використовується для того, щоб помістити повідомлення в чергу, асоційовану з ідентифікатором черги msqid. При успішному завершенні системного виклику повертається 0. У випадку помилки повертається -1, а змінній errno присвоюється код помилки.

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

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

struct msgbuf
{
   long mtype;       // Message type
   char mtext [1];   // Message text
}
```

Вона являє собою деякий шаблон структури повідомлення користувача. Повідомлення користувача - це структура, перший елемент якої обов'язково має тип long і містить тип повідомлення, а далі – інформативна частина теоретично довільної довжини (практично в Linux вона обмежена розміром 4080 байт і може бути ще зменшена системним адміністратором), що містить власне суть повідомлення. При цьому інформація зовсім не зобов'язана бути текстової, наприклад:

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

Тип повідомлення повинен бути строго позитивним числом. Дійсна довжина корисної частини інформації (тобто інформації, розташованої в структурі після типу повідомлення) повинна бути передана системному виклику як параметр msgsz. Цей параметр може бути рівним і 0, якщо вся корисна інформація полягає в самому факті наявності повідомлення. Системний виклик копіює повідомлення, розташоване за адресою, на яку вказує параметр msgp, у чергу повідомлень, задану дескриптором msqid.

Параметр flag може приймати два значення: 0 іIPC_NOWAIT. Якщо значення прапора дорівнює 0, і в черзі не вистачає місця для того, щоб помістити повідомлення, то системний виклик блокується доти, поки не звільниться місце. При значенні прапора IPC_NOWAIT системний виклик не блокується, а констатує виникнення помилки з встановленням значення змінної errno рівним EAGAIN.

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

2.4. Системний виклик msgrcv.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv (msqid, msgp, msgsz, msgtyp,
msgflg)
            int msqid;
            struct msgbuf* msgp;
            long msgtyp;
             int msgsz;
             int msgflg;
```

Системний виклик msgrcv одержує повідомлення з черги, асоційованої з ідентифікатором черги msqid, і поміщає його в структуру, на яку вказує аргумент msgp. Поле mtype містить тип одержуваного повідомлення. Поле mtext містить текст повідомлення.

Аргумент msgsz специфікує довжину прийнятого повідомлення.

Аргумент msgtyp використовується для вибору типу вибірки повідомлення з черги:

– – Якщо значення аргументу дорівнює нулю, запитується перше повідомлення в черзі.
– – Якщо більше нуля - перше повідомлення типу msgtyp.
– – Якщо менше нуля - перше повідомлення найменшого з типів, що не перевершують абсолютної величини аргументу msgtyp.

Аргумент msgflg дозволяє специфікувати виконання над повідомленням "операції з блокуванням"; для цього повинен бути скинутий прапорець IPC_NOWAIT (msgflg & IPC_NOWAIT = 0). Блокування має місце, якщо в черзі повідомлень немає повідомлення з запитуваним типом (msgtyp). Якщо прапорець IPC_NOWAIT встановлено і в черзі немає повідомлення необхідного типу, системний виклик негайно завершується невдачею.

При успішному завершенні системного виклику повертається кількість байт, дійсно поміщених у полемtext. У випадку помилки повертається -1, а змінній errno присвоюється код помилки.

2.5. Системний виклик msgctl.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
            int msqid;
            int cmd;
             struct msqid_ds* buf;
```

Системний виклик msgctl дозволяє виконувати операції керування чергою повідомлень.

Як аргумент msqid повинний виступати ідентифікатор черги повідомлень, попередньо отриманий за допомогою системного виклику msgget.

Операція визначається значенням аргументу cmd, що повинне бути одним з наступних:

IPC_STAT	Помістити поточне значення кожного поля структури даних, асоційованої з ідентифікатором черги повідомленьmsqid, у структуру, на яку вказує buf.
IPC_SET	У структурі даних, асоційованій з ідентифікатором msqid,

	перевстановити значення діючих ідентифікаторів користувача (`msg_perm.uid`) і групи (`msg_perm.gid`), прав на операції (`msg_perm.mode`) та максимально припустимого числа байт у черзі (`msg_qbytes`).
`IPC_RMI`	Видалити із системи ідентифікатор черги повідомлень `msqid`, ліквідувати чергу повідомлень і асоційовану з нею структуру даних.

Щоб виконати керуючу дію `IPC_SET` чи `IPC_RMI`, процес повинен мати діючий ідентифікатор користувача, рівний ідентифікаторам творця чи власника черги, або ідентифікатору суперкористувача. Щоб виконати дію `IPC_STAT`, потрібно право на читання.

При успішному завершенні результат дорівнює 0; у випадку помилки повертається -1, а змінній `errno` присвоюється код помилки.