**Програмні канали *(pipe*)**

Найбільш простим способом для передачі інформації за допомогою потокової моделі між різними процесами або навіть усередині одного процесу в операційній системі UNIX є ***pipe*** (канал, труба, конвеєр). Важлива відмінність pipe'а від файлу полягає в тому, що прочитана інформація негайно видаляється з нього і не може бути прочитана повторно. ***Pipe*** можна уявити собі у вигляді труби обмеженою ємності, розташованої усередині адресного простору операційної системи, доступ до вхідного і вихідного отвору якої здійснюється за допомогою системних викликів. В дійсності ***pipe*** являє собою область пам'яті, недоступну для користувача процесам безпосередньо, часто організовану у вигляді кільцевого буфера (хоча існують і інші види організації). За буферу при операціях читання та запису переміщаються два покажчика, відповідні вхідних та вихідних потоків.

При цьому вихідний покажчик ніколи не може перегнати вхідний і навпаки. Для створення нового екземпляра такого кільцевого буфера всередині операційної системи використовується системний виклик ***pipe ().***

***# include <unistd.h>***

***int pipe (int \* fd);***

Системний виклик ***pipe*** призначений для створення ***pip'а*** всередині операційної системи.

Параметр ***fd*** є покажчиком на масив з двох цілих змінних. При нормальному завершенні виклику в перший елемент масиву 0 ***fd*** \* 0 + 0 буде занесений файловий дескриптор, відповідний вихідному потоку даних pip6 а й дозволяє виконувати тільки операцію читання, а під другий елемент масиву 0 ***fd*** \* 1 + 0 буде занесений файловий дескриптор, відповідний вхідного потоку даних і дозволяє виконувати тільки операцію запису.

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

У процесі роботи системний виклик організовує виділення області пам'яті під буфер і покажчики і заносить інформацію, відповідну вхідних і вихідних потоків даних, в два елементи таблиці відкритих файлів, пов'язуючи тим самим з кожним pip6 ом два файлових дескриптора. Для одного з них дозволена тільки операція читання з ***pip6*** а, а для іншого 0 тільки операція запису в ***pipe***. Для виконання цих операцій ми можемо використовувати ті ж самі системні виклики ***read ()*** і ***write (),*** що і при роботі з файлами.

Природно, після закінчення використання вхідного або / та вихідного потоку даних,

потрібно закрити відповідний потік за допомогою системного виклику ***close ()*** для звільнення системних ресурсів. Необхідно відзначити, що, коли всі процеси, що використовують pipe, закривають всі асоційовані з ним файлові дескриптори, операційна система ліквідує pipe. Таким чином, час існування pip6 а в системі не може перевищувати час життя процесів, що працюють з ним.

Досить яскравою ілюстрацією дій по створенню ***pip'a***, записи в нього даних, читання з нього і звільненню виділених ресурсів може служити програма, організуюча роботу з pip6 ом в рамках одного процесу, наведена нижче:

***# include <sys/types.h>***

***# include <unistd.h>***

***# include <stdio.h>***

***int main () {***

***int fd [2];***

***size\_t size;***

***char string [] = "Hello, world!";***

***char resstring [14];***

***/ \* Спробуємо створити pipe \* /***

***if (pipe (fd) <0) {***

***/ \* Якщо створити pipe не вдалося, друкуємо про це повідомлення***

***і припиняємо роботу \* /***

***printf ("Can \ 't create pipe \ n");***

***exit (-1);***

***#include <sys/types.h>***

***# include <signal.h>***

***int kill (pid\_t pid, int signal);***

Послати сигнал (не маючи повноважень суперкористувача) можна тільки процесові, у якого ефективний ідентифікатор користувача збігається з ефективним ідентифікатором користувача для процесу, що посилає сигнал. Аргумент pid вказує процес, якому надсилається сигнал, а аргумент sig - який сигнал посилається. В залежності від значення аргументів:

. ***pid> 0*** сигнал посилається процесу з ідентифікатором pid;

***. pid = 0*** сигнал посилається всім процесам у групі, до якої належить посилає процес;

***pid = -1*** і посилає процес не є процесом суперкористувача, то сигнал надсилається всім процесам в системі, для яких ідентифікатор користувача збігається з ефективним ідентифікатором користувача процесу, що посилає сигнал.

. ***pid = -***1 і посилає процес є процесом суперкористувача, то сигнал надсилається всім процесам в системі, за винятком системних процесів (звичайно всім, крім процесів з ***pid = 0*** і ***pid = 1***).

. ***pid <0***, але не -1, то сигнал посилається всім процесам з групи, ідентифікатор якої дорівнює абсолютному значенню аргументу pid (якщо дозволяють привілеї).

. якщо ***sig = 0***, то проводиться перевірка на помилку, а сигнал не посилається. Це можна використовувати для перевірки правильності аргументу pid (чи є в системі процес або група процесів з відповідним ідентифікатором).

Системні виклики для установки власного обробника сигналів:

***# include <signal.h>***

***void (\* signal (int sig, void (\* handler) (int))) (int);***

***int sigaction (int sig, const struct sigaction \* act, struct sigaction \* Oldact);***

Структура sigaction має наступний формат:

***struct sigaction {***

***void (\* sa\_handler) (int);***

***void (\* sa\_sigaction) (int, siginfo\_t \*, void \*);***

***sigset\_t sa\_mask;***

***int sa\_flags;***

***void (\* sa\_restorer) (void);***

Системний виклик signal служить для зміни реакції процесу на будь-який сигнал.

Параметр ***sig*** - це номер сигналу, обробку якого належить змінити. Параметр ***handler*** описує новий спосіб обробки сигналу - це може бути покажчик на користувача функцію-обробник сигналу, спеціальне значення **SIG\_DFL** (відновити реакцію процесу на сигнал ***sig*** за замовчуванням) або спеціальне значення **SIG\_IGN** (ігнорувати надійшов сигнал ***sig***). Системний виклик повертає покажчик на старий спосіб обробки сигналу, значення якого можна використовувати для відновлення старого способу в разі потреби. Приклад користувальницької обробки сигналу ***SIGUSR1***.

***void \* my\_handler (int nsig) {код функції-обробника сигналу}***

***int main ()***

***(void) signal (SIGUSR1, my\_handler);***

Системний виклик ***sigaction*** використовується для зміни дій процесу при отриманні відповідного сигналу. Параметр ***sig*** задає номер сигналу і може бути дорівнює кожному номеру. Якщо параметр act не дорівнює нулю, то нову дію, пов'язується з сигналом ***sig***, встановлюється відповідно ***act***. Якщо ***oldac***t не дорівнює нулю, то попереднє дію записується в ***oldact***.

**5.3**