***/ \* Пробуємо записати в pipe 14 байт з нашого масиву, тобто всю рядок "Hello, world!" разом з ознакою кінця рядка \* /***

***size = write (fd [1], string, 14);***

***if (size! = 14) {***

***/ \* Якщо записалося меншу кількість байт, повідомляємо про помилку \* /***

***printf ("Can \ 't write all string \ n");***

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

***/ \* Пробуємо прочитати з pip'а 14 байт в інший масив, тобто всю записану рядок \* /***

***size = read (fd [0], resstring, 14);***

***if (size <0) {***

***/ \* Якщо прочитати не змогли, повідомляємо про помилку \* /***

***printf ("Can \ 't read string \ n");***

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

***/ \* Друкуємо прочитаний рядок \* /***

***printf ("% s \ n", resstring);***

***/ \* Закриваємо вхідний потік \* /i***

***if (close (fd [0]) <0) {***

***printf ("Can \ 't close input stream \ n");***

***}***

***/ \* Закриваємо вихідний потік \* /***

***if (close (fd [1]) <0) {***

***printf ("Can \ 't close output stream \ n");***

***}***

***return 0;***

***}***

Зрозуміло, що якби все гідність ***pip'ов*** зводилося до заміни функції копіювання з пам'яті в пам'ять усередині одного процесу на пересилку інформації через операційну систему, то овчинка не вартувала б вичинки. Однак таблиця відкритих файлів успадковується процесом-дитиною при породженні нового процесу системним викликом ***fork ()*** і входить до складу незмінної частини системного контексту процесу при системному виклику ***exec ()*** (за винятком тих потоків даних, для файлових дескрипторів яких був спеціальними засобами виставлений ознака, що спонукає операційну систему закрити їх при виконанні ***exec ().***). Ця обставина дозволяє організувати передачу інформації через **pipe** між спорідненими процесами, що мають загального прабатька, який створив **pipe**. Давайте розглянемо програму, що здійснює односпрямований зв'язок між процесом-батьком і процесом-дитиною:

***#include <sys/types.h>  
# include <unistd.h>  
# Include <stdio.h>  
int main () {  
int fd [2], result;  
size\_t size;  
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);  
}  
/ \* Породжує новий процес \* /  
result = fork ();  
if (result) {  
/ \* Якщо створити процес не вдалося, повідомляємо про це і  
завершуємо роботу \* /  
printf ("Can \ 't fork child \ n");  
exit (-1);***

***} Else if (result> 0) {  
/ \* Ми знаходимося в батьківському процесі, який будепередавати інформацію процесові-дитині. У цьому процесівихідний потік даних нам не знадобиться, тому закриваємо його. \* /  
close (fd [0]);  
/ \* Пробуємо записати в pipe 14 байт, тобто весь рядок "Hello, world!" разом з ознакою кінця рядка \* /  
size = write (fd [1], "Hello, world!", 14);  
if (size! = 14) {  
/ \* Якщо записалося меншу кількість байт, повідомляємо про помилку і завершуємо роботу \* /  
printf ("Can \ 't write all string \ n");  
exit (-1);  
}  
/ \* Закриваємо вхідний потік даних, на цьому батько припиняє роботу \* /  
close (fd [1]);  
printf ("Parent exit \ n");  
} Else {  
/ \* Ми знаходимося в породженому процесі, який буде отримувати інформацію від процесу-батька. Він успадкував від батька таблицю відкритих файлів і, знаючи файлові дескриптори, відповідні pip, іможет його використовувати. У цьому процесі вхідний потік даних нам не іпонадобітся, тому закриваємо його. \* /  
close (fd [1]);  
/ \* Пробуємо прочитати з pip'а 14 байт в масив, тобто всю  
записану рядок \* /  
size = read (fd [0], resstring, 14);  
if (size <0) {  
/ \* Якщо прочитати не змогли, повідомляємо про помилку і завершуємо роботу \* /  
printf ("Can \ 't read string \ n");  
exit (-1);  
}  
/ \* Друкуємо прочитану рядок \* /  
printf ("% s \ n", resstring);  
/ \* Закриваємо вхідний потік і завершуємо роботу \* /  
close (fd [0]);  
}  
return 0;  
}***

***Pipe*** служить для організації односпрямованої або симплексного зв'язку. Якби в попередньому прикладі ми спробували організувати через pipe двосторонній зв'язок, коли процес-батько пише інформацію в ***pipe***, припускаючи, що її отримає процес-дитина, а потім читає інформацію з pip6 а, припускаючи, що її записав породжений процес, то могла б виникнути ситуація, в якій процес-батько прочитав би власну інформацію, а процес-дитина не отримала б нічого. Для використання одного pip6 а в двох напрямках необхідні спеціальні засоби синхронізації процесів. Більш простий спосіб організації двонаправленої зв'язку між спорідненими процесами полягає у використанні двох pipe.

Будьте уважні при написанні програм, які обмінюються великими об'ємами інформації через pipe. Пам'ятайте, що за один раз з ***pip6*** а може прочитати менше інформації, ніж ви запитували, і за один раз в ***pipe*** може записатися менше інформації, ніж вам хотілося б. Перевіряйте значення, що повертаються викликами!

Одна з особливостей поведінки блокує системного виклику ***rea ()*** пов'язана з спробою читання з пустого ***pip'а***. Якщо є процеси, у яких цей pipe відкритий для запису, то системний виклик блокується і чекає появи інформації. Якщо таких процесів немає, він поверне значення 0 без блокування процесу. Ця особливість призводить до необхідності закриття файлового дескриптора, асоційованого з вхідним кінцем ***pip'a***, Аналогічної особливістю поведінки при відсутності процесів, у яких pipe відкритий для читання, володіє і системний виклик ***write(),*** з чим пов'язана необхідність закриття файлового дескриптора, асоційованого з вихідним кінцем ***pip'a***, в процесі, який буде використовувати ***pipe*** для запису ***(close (fd \* 0 +)*** в процесі- атькові в тій же програмі).

**5.4**