Potoki

Systemy Operacyjne 2

Piotr Zierhoffer

6 listopada 2011

Łącza

W systemach uniksowych dwa rodzaje łączy:

- łącze nienazwane — potok
  - istnieje tak długo, jak jest otwarte
  - po zamknięciu wszystkich deskryptorów i–node jest zwalniany
- łącze nazwane — kolejka FIFO
  - posiada nazwę :)
  - istnieje jako plik
  - po zamknięciu deskryptorów zwalniane są bloki danych
- jeżeli proces posiada deskryptor łącza, to komunikacja przebiega tak samo w obu przypadkach

Komunikacja

Łącza posiadają podobny do plików interface dostępu:

- funkcje `read`/`write`
- dane odczytywane są w takiej kolejności, w jakiej są zapisywane
- dane odczytane z łącza są usuwane
- wywołania funkcji są blokujące
- dostęp jest sekwencyjny
    - na łączu nie można wykonywać operacji przesunięcia wskaźnika (`lseek`, przesuwanie przez `fcntl`)

Procesy mogą komunikować się przez potok gdy:

- mają wspólnego przodka
- jeden jest przodkiem drugiego

Otwarcie potoku

- Funkcja:
  `int pipe(int pdes[2])`
- Parametry:
  - `pdes[2]` — para sprzężonych deskryptorów pliku, wskazujących na i–node potoku
    - `pdes[0]` — deskryptor do odczytu
    - `pdes[1]` — deskryptor do zapisu

Uwaga!

Parametr wyjściowy!

- Wartość zwracana:
  - 0 lub -1 w wypadku błędu

Zamykanie potoku

- Funkcja
  `int close(int fd)`

- Uwagi dla deskryptora zapisu:
  - jeżeli istnieją inne procesy mający otwarty potok do zapisu, nic się nie dzieje
  - gdy nie ma więcej procesów piszących a potok jest pusty, procesy, które czekały na odczyt z potoku, zostają obudzone i ich funkcje `read` zwracają 0

- Uwagi dla deskryptora odczytu
  - jeżeli istnieją inne procesy mające potok otwarty do odczytu, nic się nie dzieje
  - gdy żaden proces nie czyta, do wszystkich procesów czekających na zapis wysyłany jest sygnał `SIGPIPE`

Pisanie do potoku

- Funkcja
  `int read( inf fd, void *buf, size_t count )`

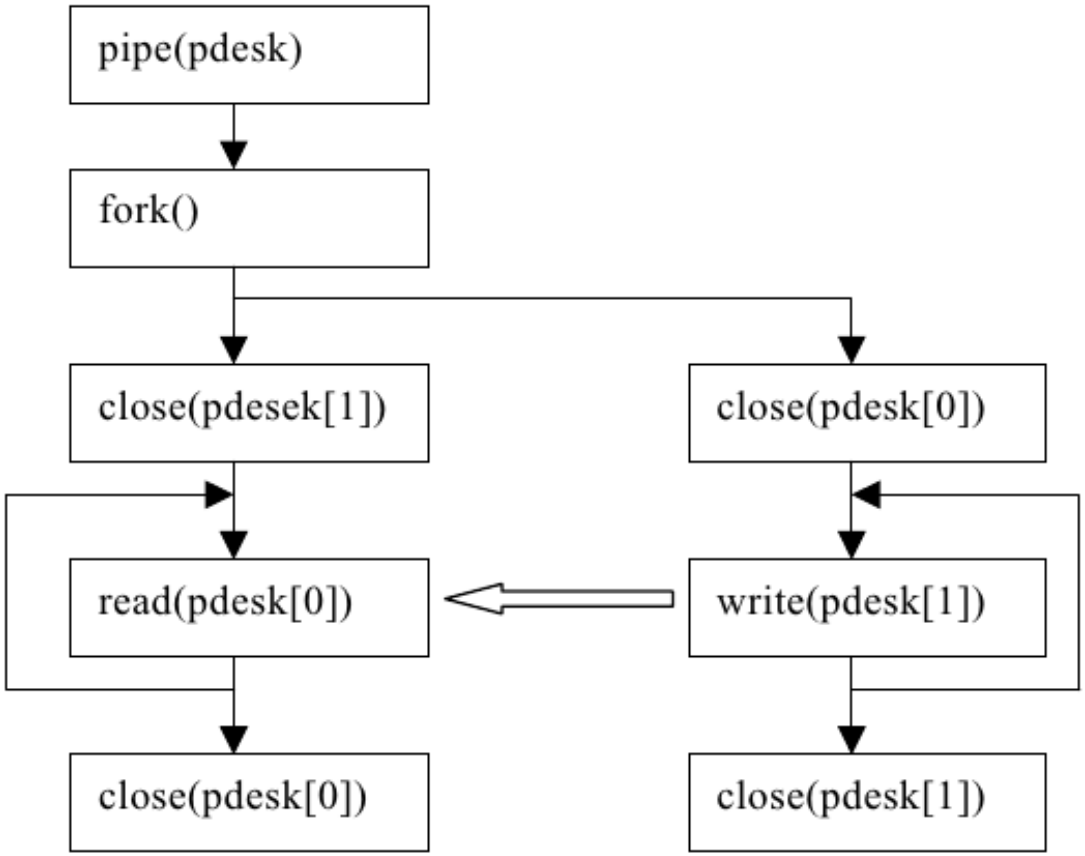
- Uwagi:
  - odczytane dane są usuwane z potoku!
  - jeżeli w potoku nie ma danych, ale istnieją otwarte deskryptory do zapisu do tego potoku, funkcja będzie oczekiwać na dane
    - chyba, że ustawiono flagę `O_NDELAY` przez `fcntl`
  - 0 zwrócone dopiero po zamknięciu **wszystkich** deskryptorów do zapisu
  - funkcja odczytuje tyle danych, ile jest aktualnie dostępne
    - jeżeli przeczyta $<$ count, nie oznacza to końca nadawania

Czytanie z potoku

- Funkcja
  `int write( inf fd, void *buf, size_t count )`

- Uwagi:
  - zapis do potoku wykonywany jest atomowo
    - dane się nie przeplatają!
  - potok ma ograniczoną pojemność
    - `fpathconf(pdesk[0], _PC_PIPE_BUF);`
  - zapis zostaje wstrzymany, aż zwolni się dostateczna ilość miejsca w potoku
    - chyba, że ustawiono flagę `O_NDELAY` przez `fcntl`

Schemat pracy



Przeadresowanie wejścia/wyjścia

- Problem: jak przekierować wyjcie jednego polecenia na wejście drugiego?

```
int pd[2];
pipe(pd);
if(fork()){
    execlp("ls", "ls", NULL);
} else {
    execlp("wc", "wc", NULL);
}
```

Przeadresowanie wejścia/wyjścia, cd

- Funkcja:
  `int dup(int oldfd)`
  `int dup2(int oldfd, int newfd)`

- Zastosowanie:
  - `dup2(pd[1], 1);`
    powiel OTWARTY deskryptor do zapisu `pd[1]`, nowemu nadaj numer 1 — standardowe wyjście

```
close(pd[0]);
dup2(pd[1],1);
execlp("ls", "ls", NULL);
```