Potoki (łącza nienazwane)

Systemy Operacyjne 2 — laboratorium

Mateusz Hołenko

15 października 2011

Plan zajęć

1. Łącza w systemie Linux
   - potoki vs. kolejki FIFO
   - specyfika łączy nienazwanych
   - schemat komunikacji przez łącza
   - problemy
2. Funkcje systemowe
   - tworzenie łącza
   - odczyt z łącza
   - zapis do łącza
   - zamykanie łącza
3. Zadania praktyczne

Łącza w systemie Linux

Łącza nazwane i nienazwane

- łącza służą do komunikacji między procesami w systemie Linux
- wyróżnia się dwa rodzaje łączy:
  - nienazwane, inaczej potoki
  - nazwane, inaczej kolejki FIFO
- łącza są implementowane jako pliki specjalne
  - posiadają swój i-węzeł
  - posiadają bloki z danymi
  - można do nich pisać i z nich czytać

Łącza nazwane i nienazwane

- łącza służą do komunikacji między procesami w systemie Linux
- wyróżnia się dwa rodzaje łączy:
  - nienazwane, inaczej potoki
  - nazwane, inaczej kolejki FIFO
- łącza są implementowane jako pliki specjalne
  - posiadają swój i-węzeł
  - posiadają bloki z danymi
  - można do nich pisać i z nich czytać

Łącza nazwane i nienazwane

- łącza służą do komunikacji między procesami w systemie Linux
- wyróżnia się dwa rodzaje łączy:
  - nienazwane, inaczej potoki
  - nazwane, inaczej kolejki FIFO
- łącza są implementowane jako pliki specjalne
  - posiadają swój i-węzeł
  - posiadają bloki z danymi
  - można do nich pisać i z nich czytać

Ograniczenia łączy

- limit wielkości
- dostęp jedynie sekwencyjny (brak możliwości wykorzystania funkcji `lseek`)
- specyfika odczytu danych
  - odczytywane dane są usuwane z łącza
  - dane odczytywane są w takiej samej kolejności jak były zapisywane
- blokowanie operacji
  - zapisu — gdy łącze jest pełne
  - odczytu — gdy łącze jest puste, a otwarty jest jakiś deskryptor do zapisu

Ograniczenia łączy

- limit wielkości
- dostęp jedynie sekwencyjny (brak możliwości wykorzystania funkcji `lseek`)
- specyfika odczytu danych
  - odczytywane dane są usuwane z łącza
  - dane odczytywane są w takiej samej kolejności jak były zapisywane
- blokowanie operacji
  - zapisu — gdy łącze jest pełne
  - odczytu — gdy łącze jest puste, a otwarty jest jakiś deskryptor do zapisu

Ograniczenia łączy

- limit wielkości
- dostęp jedynie sekwencyjny (brak możliwości wykorzystania funkcji `lseek`)
- specyfika odczytu danych
  - odczytywane dane są usuwane z łącza
  - dane odczytywane są w takiej samej kolejności jak były zapisywane
- blokowanie operacji
  - zapisu — gdy łącze jest pełne
  - odczytu — gdy łącze jest puste, a otwarty jest jakiś deskryptor do zapisu

Ograniczenia łączy

- limit wielkości
- dostęp jedynie sekwencyjny (brak możliwości wykorzystania funkcji `lseek`)
- specyfika odczytu danych
  - odczytywane dane są usuwane z łącza
  - dane odczytywane są w takiej samej kolejności jak były zapisywane
- blokowanie operacji
  - zapisu — gdy łącze jest pełne
  - odczytu — gdy łącze jest puste, a otwarty jest jakiś deskryptor do zapisu

Łącza nienazwane

- nie są widoczne w systemie plików
- istnieją tak długo, jak otwarty jest jakikolwiek deskryptor do tego łącza
- wykorzystywane mogą być przez procesy znające deskryptor potoku
- identyfikowane są przez dwa deskryptory — do odczytu oraz do zapisu

Ważne!

Potoki wykorzystywane mogą być jedynie do komunikacji między procesami macierzystymi i potomnymi lub posiadającymi wspólnego przodka!

Łącza nienazwane

- nie są widoczne w systemie plików
- istnieją tak długo, jak otwarty jest jakikolwiek deskryptor do tego łącza
- wykorzystywane mogą być przez procesy znające deskryptor potoku
- identyfikowane są przez dwa deskryptory — do odczytu oraz do zapisu

Ważne!

Potoki wykorzystywane mogą być jedynie do komunikacji między procesami macierzystymi i potomnymi lub posiadającymi wspólnego przodka!

Łącza nienazwane

- nie są widoczne w systemie plików
- istnieją tak długo, jak otwarty jest jakikolwiek deskryptor do tego łącza
- wykorzystywane mogą być przez procesy znające deskryptor potoku
- identyfikowane są przez dwa deskryptory — do odczytu oraz do zapisu

Ważne!

Potoki wykorzystywane mogą być jedynie do komunikacji między procesami macierzystymi i potomnymi lub posiadającymi wspólnego przodka!

Łącza nienazwane

- nie są widoczne w systemie plików
- istnieją tak długo, jak otwarty jest jakikolwiek deskryptor do tego łącza
- wykorzystywane mogą być przez procesy znające deskryptor potoku
- identyfikowane są przez dwa deskryptory — do odczytu oraz do zapisu

Ważne!

Potoki wykorzystywane mogą być jedynie do komunikacji między procesami macierzystymi i potomnymi lub posiadającymi wspólnego przodka!

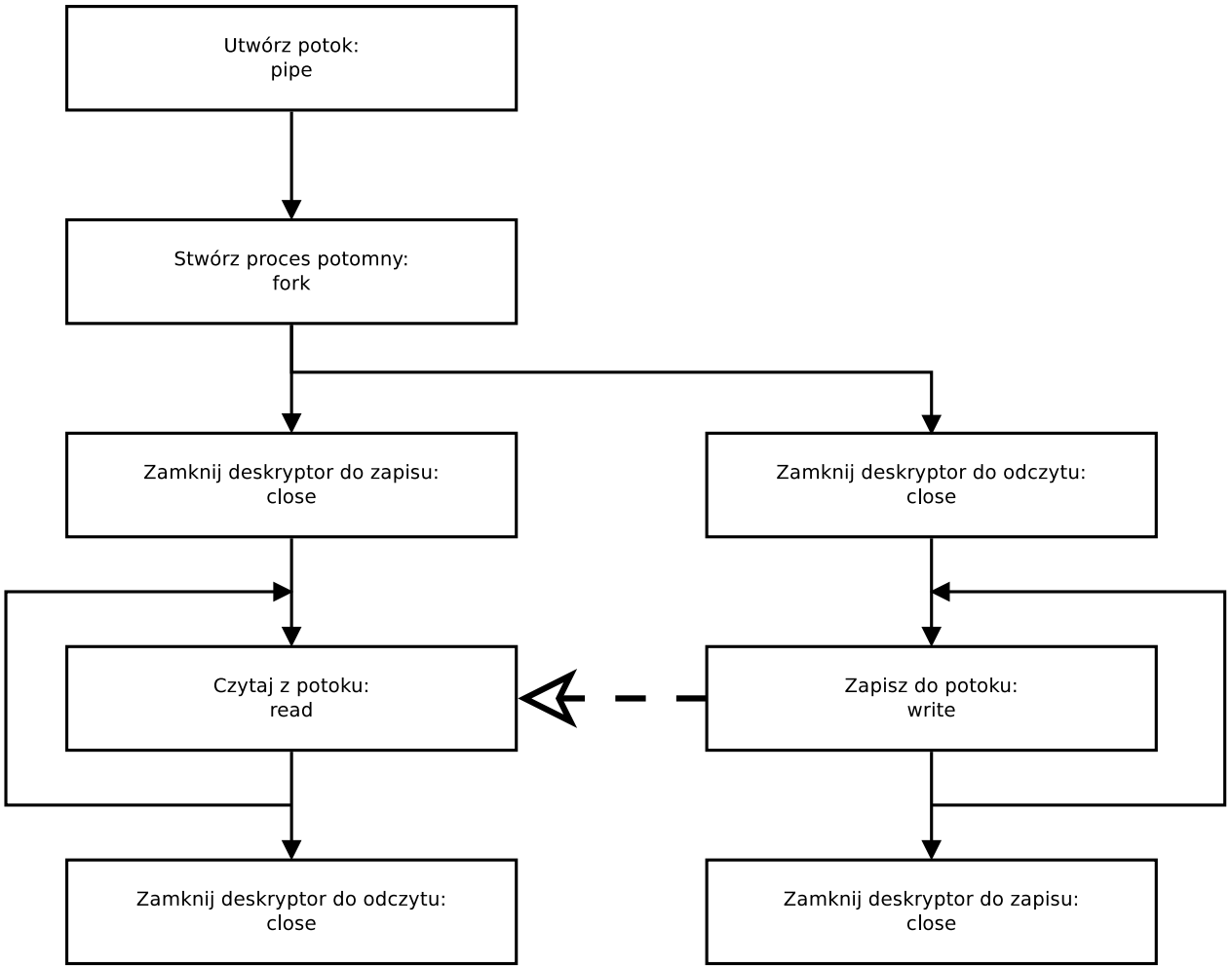
Łącza nienazwane

- nie są widoczne w systemie plików
- istnieją tak długo, jak otwarty jest jakikolwiek deskryptor do tego łącza
- wykorzystywane mogą być przez procesy znające deskryptor potoku
- identyfikowane są przez dwa deskryptory — do odczytu oraz do zapisu

Ważne!

Potoki wykorzystywane mogą być jedynie do komunikacji między procesami macierzystymi i potomnymi lub posiadającymi wspólnego przodka!

Schemat komunikacji przez potok



Rysunek: Schemat użycia potoku do komunikacji jednokierunkowej

Problemy

- funkcje `read` i `write` są blokujące
- istnieje niebezpieczeństwo zakleszczenia
- należy uważać podczas korzystania z innych mechanizmów synchronizacji (np. funkcji `wait`)

Problemy

- funkcje `read` i `write` są blokujące
- istnieje niebezpieczeństwo zakleszczenia
- należy uważać podczas korzystania z innych mechanizmów synchronizacji (np. funkcji `wait`)

Problemy

- funkcje `read` i `write` są blokujące
- istnieje niebezpieczeństwo zakleszczenia
- należy uważać podczas korzystania z innych mechanizmów synchronizacji (np. funkcji `wait`)

Funkcje systemowe

Tworzenie łącza

- tworzenie łącza nienazwanego (potoku)

```
int pipe(int pipefd[2])
```

- przykład zastosowania

```
#include <unistd.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
  int fds[2];
  char output[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
  char input[6] = { 0 };

  printf("Wartość input: %s\n", input);

  pipe(fds);
  write(fds[1], output, 6);
  read(fds[0], input, 6);

  printf("Wartość input: %s\n", input);
  return 0;
}
```

Tworzenie łącza

- tworzenie łącza nienazwanego (potoku)

```
int pipe(int pipefd[2])
```

- przykład zastosowania

```
#include <unistd.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
  int fds[2];
  char output[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
  char input[6] = { 0 };

  printf("Wartość input: %s\n", input);

  pipe(fds);
  write(fds[1], output, 6);
  read(fds[0], input, 6);

  printf("Wartość input: %s\n", input);
  return 0;
}
```

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read` (deskryptor `pipefd[0]`)
- zapis danych — funkcja `write` (deskryptor `pipefd[1]`)
- zamykanie deskryptora — standardowa funkcja `close`
- zamykanie deskryptora do odczytu
  - są inne deskryptory do odczytu — nic się nie dzieje
  - brak innych deskryptorów do odczytu — procesy oczekujące na zapis (wiszące na `write`) otrzymują sygnał SIGPIPE
- zamykanie deskryptora do zapisu
  - są inne deskryptory do zapisu — nic się nie dzieje
  - brak innych deskryptorów do zapisu — procesy oczekujące na odczyt wychodzą z funkcji `read`, która zwraca wartość 0

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read` (deskryptor `pipefd[0]`)
- zapis danych — funkcja `write` (deskryptor `pipefd[1]`)
- zamykanie deskryptora — standardowa funkcja `close`
- zamykanie deskryptora do odczytu
  - są inne deskryptory do odczytu — nic się nie dzieje
  - brak innych deskryptorów do odczytu — procesy oczekujące na zapis (wiszące na `write`) otrzymują sygnał SIGPIPE
- zamykanie deskryptora do zapisu
  - są inne deskryptory do zapisu — nic się nie dzieje
  - brak innych deskryptorów do zapisu — procesy oczekujące na odczyt wychodzą z funkcji `read`, która zwraca wartość 0

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read` (deskryptor `pipefd[0]`)
- zapis danych — funkcja `write` (deskryptor `pipefd[1]`)
- zamykanie deskryptora — standardowa funkcja `close`
- zamykanie deskryptora do odczytu
  - są inne deskryptory do odczytu — nic się nie dzieje
  - brak innych deskryptorów do odczytu — procesy oczekujące na zapis (wiszące na `write`) otrzymują sygnał SIGPIPE
- zamykanie deskryptora do zapisu
  - są inne deskryptory do zapisu — nic się nie dzieje
  - brak innych deskryptorów do zapisu — procesy oczekujące na odczyt wychodzą z funkcji `read`, która zwraca wartość 0

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read` (deskryptor `pipefd[0]`)
- zapis danych — funkcja `write` (deskryptor `pipefd[1]`)
- zamykanie deskryptora — standardowa funkcja `close`
- zamykanie deskryptora do odczytu
  - są inne deskryptory do odczytu — nic się nie dzieje
  - brak innych deskryptorów do odczytu — procesy oczekujące na zapis (wiszące na `write`) otrzymują sygnał SIGPIPE
- zamykanie deskryptora do zapisu
  - są inne deskryptory do zapisu — nic się nie dzieje
  - brak innych deskryptorów do zapisu — procesy oczekujące na odczyt wychodzą z funkcji `read`, która zwraca wartość 0

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read` (deskryptor `pipefd[0]`)
- zapis danych — funkcja `write` (deskryptor `pipefd[1]`)
- zamykanie deskryptora — standardowa funkcja `close`
- zamykanie deskryptora do odczytu
  - są inne deskryptory do odczytu — nic się nie dzieje
  - brak innych deskryptorów do odczytu — procesy oczekujące na zapis (wiszące na `write`) otrzymują sygnał SIGPIPE
- zamykanie deskryptora do zapisu
  - są inne deskryptory do zapisu — nic się nie dzieje
  - brak innych deskryptorów do zapisu — procesy oczekujące na odczyt wychodzą z funkcji `read`, która zwraca wartość 0

Zadania praktyczne

Zadania

Zadanie 1

Napisz program, który tworzy dwa procesy – pierwszy z procesów powinien zapisać do potoku napis „Hallo” a drugi proces powinien ten napis odczytać.

Zadanie 2

Zmodyfikuj program 1, tak by przez potok stworzony przez rodzica komunikowały się dwa procesy potomne.

Zadanie 3

Napisz program tworzący trzy procesy, z których dwa zapisują do potoku, a trzeci odczytuje z niego i drukuje otrzymane komunikaty.

Zadania

Zadanie 1

Napisz program, który tworzy dwa procesy – pierwszy z procesów powinien zapisać do potoku napis „Hallo” a drugi proces powinien ten napis odczytać.

Zadanie 2

Zmodyfikuj program 1, tak by przez potok stworzony przez rodzica komunikowały się dwa procesy potomne.

Zadanie 3

Napisz program tworzący trzy procesy, z których dwa zapisują do potoku, a trzeci odczytuje z niego i drukuje otrzymane komunikaty.

Zadania

Zadanie 1

Napisz program, który tworzy dwa procesy – pierwszy z procesów powinien zapisać do potoku napis „Hallo” a drugi proces powinien ten napis odczytać.

Zadanie 2

Zmodyfikuj program 1, tak by przez potok stworzony przez rodzica komunikowały się dwa procesy potomne.

Zadanie 3

Napisz program tworzący trzy procesy, z których dwa zapisują do potoku, a trzeci odczytuje z niego i drukuje otrzymane komunikaty.

Zadania

Zadanie 4

Napisz program realizujący potok `ls | wc`.

Zadanie 5

Napisz program konwertujący wynik polecenia `ls` z małych liter na duże.

Zadanie 6

Napisz program będący odpowiednikiem programu 5, realizujący programowo potok `ls |tr a-z A-Z`.

Zadania

Zadanie 4

Napisz program realizujący potok `ls | wc`.

Zadanie 5

Napisz program konwertujący wynik polecenia `ls` z małych liter na duże.

Zadanie 6

Napisz program będący odpowiednikiem programu 5, realizujący programowo potok `ls |tr a-z A-Z`.

Zadania

Zadanie 4

Napisz program realizujący potok `ls | wc`.

Zadanie 5

Napisz program konwertujący wynik polecenia `ls` z małych liter na duże.

Zadanie 6

Napisz program będący odpowiednikiem programu 5, realizujący programowo potok `ls |tr a-z A-Z`.

Zadania

Zadanie 7

Zrealizuj następujące potoki:

- `finger | cut -d' ' -f1`
- `ls -l | grep ^d | more`
- `ps -ef | tr -s ' ' : | cut -d: -f1 | sort | uniq -c | sort n`
- `cat /etc/group | head -5 > grupy.txt`