Kolejki FIFO (łącza nazwane)

Systemy Operacyjne 2 — laboratorium

Mateusz Hołenko

6 listopada 2011

Plan zajęć

1. Łącza w systemie Linux
   - kolejki FIFO vs. potoki
   - specyfika łączy nazwanych
   - schemat komunikacji przez łącza
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
  - otwarcia — gdy łącze nie zostało otwarte komplementarnie

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
  - otwarcia — gdy łącze nie zostało otwarte komplementarnie

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
  - otwarcia — gdy łącze nie zostało otwarte komplementarnie

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
  - otwarcia — gdy łącze nie zostało otwarte komplementarnie

Łącza nazwane

- są widoczne w systemie plików jako pliki specjalne
- mogą być tworzone za pomocą funkcji systemowych lub w konsoli (polecenie `mkfifo`)
- wykorzystywane mogą być przez dowolne procesy mające dostęp do pliku specjalnego łącza

Ważne!

W odróżnieniu od łączy nienazwanych, kolejki FIFO mogą być wykorzystywane do komunikacji między dowolnymi procesami.

Łącza nazwane

- są widoczne w systemie plików jako pliki specjalne
- mogą być tworzone za pomocą funkcji systemowych lub w konsoli (polecenie `mkfifo`)
- wykorzystywane mogą być przez dowolne procesy mające dostęp do pliku specjalnego łącza

Ważne!

W odróżnieniu od łączy nienazwanych, kolejki FIFO mogą być wykorzystywane do komunikacji między dowolnymi procesami.

Łącza nazwane

- są widoczne w systemie plików jako pliki specjalne
- mogą być tworzone za pomocą funkcji systemowych lub w konsoli (polecenie `mkfifo`)
- wykorzystywane mogą być przez dowolne procesy mające dostęp do pliku specjalnego łącza

Ważne!

W odróżnieniu od łączy nienazwanych, kolejki FIFO mogą być wykorzystywane do komunikacji między dowolnymi procesami.

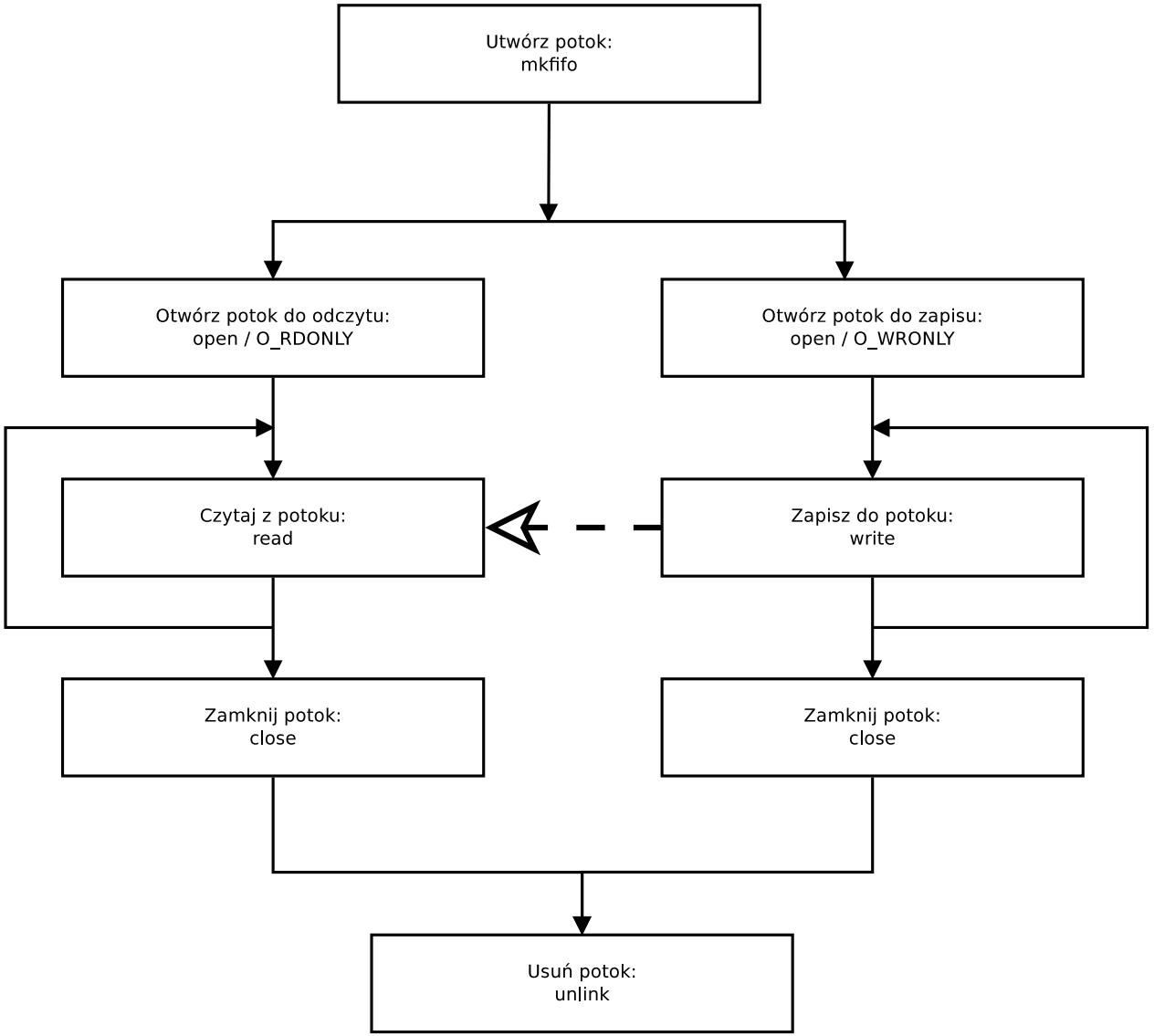
Łącza nazwane

- są widoczne w systemie plików jako pliki specjalne
- mogą być tworzone za pomocą funkcji systemowych lub w konsoli (polecenie `mkfifo`)
- wykorzystywane mogą być przez dowolne procesy mające dostęp do pliku specjalnego łącza

Ważne!

W odróżnieniu od łączy nienazwanych, kolejki FIFO mogą być wykorzystywane do komunikacji między dowolnymi procesami.

Schemat komunikacji przez potok



Rysunek: Schemat użycia potoku nazwanego

Problemy

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

Problemy

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

Problemy

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

Funkcje systemowe

Tworzenie łącza

- tworzenie łącza nazwanego (kolejki FIFO)
  - `pathname` — ścieżka do pliku specjalnego łącza
  - `mode` — prawa dostępu do pliku

```
int mkfifo(const char *pathname, mode_t mode)
```

- otwieranie łącza
  - `pathname` — ścieżka do pliku specjalnego łącza
  - `flags` — tryb otwarcia łącza: O_RDONLY lub O_WRONLY

```
int open(const char *pathname, int flags)
```

Tworzenie łącza

- tworzenie łącza nazwanego (kolejki FIFO)
  - `pathname` — ścieżka do pliku specjalnego łącza
  - `mode` — prawa dostępu do pliku

```
int mkfifo(const char *pathname, mode_t mode)
```

- otwieranie łącza
  - `pathname` — ścieżka do pliku specjalnego łącza
  - `flags` — tryb otwarcia łącza: O_RDONLY lub O_WRONLY

```
int open(const char *pathname, int flags)
```

Tworzenie łącza

Ważne!

Łącze musi być otwarte dwukrotnie — raz z flagą O_RDONLY oraz drugi raz z flagą O_WRONLY. Do tego czasu proces wywołujący funkcję open zostaje zawieszony!

Odczyt i zapis danych, zamykanie łącza

- odczyt danych — funkcja `read`
- zapis danych — funkcja `write`
- 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`
- zapis danych — funkcja `write`
- 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`
- zapis danych — funkcja `write`
- 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`
- zapis danych — funkcja `write`
- 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`
- zapis danych — funkcja `write`
- 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

Napisać program który tworzy dwa procesy — zapisujący do kolejki FIFO i odczytujący z niej napis „HALLO!”.

Zadanie 2

Zrealizować następujące potoki: (1) `ls | wc` oraz (2) `ps -ef | tr -s ‘ ‘ :| cut -d: -f1 |sort| uniq -c |sort -n`

Zadanie 3

Napisać program tworzący dwa procesy: klienta i serwera. Serwer tworzy ogólnodostępną kolejkę FIFO, i czeka na zgłoszenia klientów. Każdy klient tworzy własną kolejkę, poprzez którą będą przychodzić odpowiedzi serwera. Zadaniem klienta jest przesłanie nazwy stworzonej przez niego kolejki, a serwera odesłaniem poprzez kolejkę stworzoną przez klienta wyniku polecenia `ls`.

Zadania

Zadanie 1

Napisać program który tworzy dwa procesy — zapisujący do kolejki FIFO i odczytujący z niej napis „HALLO!”.

Zadanie 2

Zrealizować następujące potoki: (1) `ls | wc` oraz (2) `ps -ef | tr -s ‘ ‘ :| cut -d: -f1 |sort| uniq -c |sort -n`

Zadanie 3

Napisać program tworzący dwa procesy: klienta i serwera. Serwer tworzy ogólnodostępną kolejkę FIFO, i czeka na zgłoszenia klientów. Każdy klient tworzy własną kolejkę, poprzez którą będą przychodzić odpowiedzi serwera. Zadaniem klienta jest przesłanie nazwy stworzonej przez niego kolejki, a serwera odesłaniem poprzez kolejkę stworzoną przez klienta wyniku polecenia `ls`.

Zadania

Zadanie 1

Napisać program który tworzy dwa procesy — zapisujący do kolejki FIFO i odczytujący z niej napis „HALLO!”.

Zadanie 2

Zrealizować następujące potoki: (1) `ls | wc` oraz (2) `ps -ef | tr -s ‘ ‘ :| cut -d:  -f1 |sort| uniq -c |sort -n`

Zadanie 3

Napisać program tworzący dwa procesy: klienta i serwera. Serwer tworzy ogólnodostępną kolejkę FIFO, i czeka na zgłoszenia klientów. Każdy klient tworzy własną kolejkę, poprzez którą będą przychodzić odpowiedzi serwera. Zadaniem klienta jest przesłanie nazwy stworzonej przez niego kolejki, a serwera odesłaniem poprzez kolejkę stworzoną przez klienta wyniku polecenia `ls`.

Zadania

Zadanie 4

Zmodyfikować program 3, tak, by kolejka utworzona przez klienta była dwukierunkowa, klient publiczną kolejką powinien przesyłać nazwę stworzonej przez siebie kolejki. Dalsza wymiana komunikatów powinna odbywać się poprzez kolejkę stworzoną przez klienta. Klient kolejką tą powinien wysyłać polecenia, zadaniem serwera jest wykonywanie tych poleceń i odsyłanie wyników.

Zadanie 5

Napisać program tworzący pierścień procesów połączonych za pomocą kolejki FIFO. Liczbę procesów w pierścieniu podaje użytkownik. Jeden z procesów ma zainicjalizować wysłanie komunikatu, a pozostałe odczytać go, wyświetlić na ekranie i wysłać dalej, dopisując do komunikatu własny identyfikator.

Zadania

Zadanie 4

Zmodyfikować program 3, tak, by kolejka utworzona przez klienta była dwukierunkowa, klient publiczną kolejką powinien przesyłać nazwę stworzonej przez siebie kolejki. Dalsza wymiana komunikatów powinna odbywać się poprzez kolejkę stworzoną przez klienta. Klient kolejką tą powinien wysyłać polecenia, zadaniem serwera jest wykonywanie tych poleceń i odsyłanie wyników.

Zadanie 5

Napisać program tworzący pierścień procesów połączonych za pomocą kolejki FIFO. Liczbę procesów w pierścieniu podaje użytkownik. Jeden z procesów ma zainicjalizować wysłanie komunikatu, a pozostałe odczytać go, wyświetlić na ekranie i wysłać dalej, dopisując do komunikatu własny identyfikator.