Systemy operacyjne

Laboratorium 5

Mateusz Małek
7 kwietnia 2017

Laboratorium 5

Potoki nienazwane i nazwane

Uwaga, kolokwium!

28 kwietnia 2017

(w trakcie normalnych zajęć naszej grupy)

Potoki (ang. pipes)

- Zamiast “potoki” wolałbym nazwę “rury”
- Jeden z podstawowych mechanizmów IPC
  - Elastyczniejsze niż sygnały
  - Wydajniejsze niż komunikacja oparta o zapisywanie do plików
- Potoki po części wyglądają jak pliki - można do nich pisać, można z nich czytać, nie można jednak zmieniać położenia kursora pliku (czyli nie można korzystać z seek)

Korzystanie z potoków

- Używamy funkcji read i write w odniesieniu do deskryptora związanego z potokiem
- Jeżeli czytamy z potoku który nie jest gdziekolwiek otwarty do zapisu, read natychmiast zwraca EOF
- Jeżeli czytamy z potoku w którym nie ma danych, read blokuje się w oczekiwaniu aż się pojawią (zwróci niekoniecznie tyle ile zażądaliśmy!)
- Jeżeli piszemy do potoku którego bufor jest pełny, write blokuje się do momentu aż część danych zostanie odczytana
- Jeżeli wszystkie procesy czytające z potoku zakończą się/zamkną deskryptor do odczytu, próba zapisu zakończy się otrzymaniem SIGPIPE, a w przypadku jego zignorowania - z write zostanie zwrócony błąd EPIPE
- Nieużywany potok (deskryptor do odczytu/zapisu) zamykamy przy użyciu funkcji close

Potoki nienazwane

- Pozwalają na komunikację procesów spokrewnionych (np. rodzic i dziecko, dziecko i rodzic, przy odpowiednim użyciu również “rodzeństwa” - dzieci tego samego rodzica)
- Tworzone funkcją int pipe(int pipefd[2]) - man 2 pipe
  - Należy przekazać wskaźnik do tablicy dwuelementowej (czyli po prostu nazwę zmiennej z taką tablicą)
  - Z pipefd[0] możemy odczytywać dane
  - Do pipefd[1] możemy zapisywać dane
- Typowe użycie: w procesie macierzystym tworzymy potok funkcją pipe, wykonujemy fork, w procesie potomnym zamykamy pipefd[1] i czytamy z pipefd[0], a w procesie macierzystym zamykamy pipefd[0] i piszemy do pipefd[1] (ew. zamieniamy miejscami proces macierzysty i potomny lub używamy innego potomka zamiast procesu macierzystego)

A jak połączyć stdin i stdout różnych procesów…?

- Można użyć funkcji int dup2(int oldfd, int newfd) - man 2 dup2
  - oldfd to numer deskryptora, który chcemy wykorzystać
  - newfd to numer deskryptora, który chcemy podmienić
  - Innymi słowy:
    - Aby koniec potoku (ten, z którego wychodzą dane) jako standardowe wejście do procesu, należy wywołać:
      dup2(pipefd[0], STDIN_FILENO)
    - Aby dane wypisywane przez proces na standardowe wyjście były zapisywane do potoku, należy wywołać:
      dup2(pipefd[1], STDOUT_FILENO)

Dla wygody...

- FILE* popen(const char* command, const char* type) - man 3 popen
  - Jako command podajemy polecenie w formie takiej, jaką podalibyśmy w terminalu (podajemy po prostu linię tekstu - w tym nazwa programu, ewentualna ścieżka, argumenty…)
  - Jako type podajemy "r" (gdy chcemy czytać stdout komendy) lub "w" (gdy chcemy zapisywać na stdin komendy)
  - Zwrócony uchwyt używamy z funkcją fread albo fwrite (w zależności od wybranego mode)
- int pclose(FILE* stream) - man 3 pclose
  - Czeka na zakończenie procesu związanego z przekazanym uchwytem i zwraca jego status wyjścia (albo ustawi errno, jeśli jego uzyskanie było niemożliwe)

Potoki nazwane

- Pozwalają na komunikację także procesów niespokrewnionych (ale nadal w obrębie pojedynczej maszyny)
- Mają reprezentację w postaci specjalnego pliku (ścieżki) na dysku
- Dane zapisywane do potoku nazwanego nie są zapisywane na dysku, plik pełni wyłącznie rolę ”adresu” potoku
- Tworzone funkcją int mkfifo(const char *pathname, mode_t mode) - man 3 mkfifo
- Można również utworzyć komendą mkfifo w terminalu (man 1 mkfifo)
- Alternatywnie: funkcja systemowa mknod (man 2 mknod) i komenda mknod w terminalu (man 1 mknod)

Potoki nazwane

- Możemy mieć wiele procesów piszących i czytających
- W praktyce: wiele procesów piszących i nie więcej niż jeden czytający
- Zapis nie więcej niż PIPE_BUF jest atomowy

Dziękuję za uwagę