Spis treści

Na podstawie:

- *M.J.Rochkind - Programowanie w systemie UNIX dla zaawansowanych (Advanced UNIX Programming)*
- `https://ai.ia.agh.edu.pl/wiki/pl:dydaktyka:so:2017:labs:lab_intro`
- *Graham Glass, King Ables - Linux dla programistów i użytkowników*

1 Pojęcia podstawowe potrzebne do zrozumienia tematu

Komunikacja międzyprocesowa
Jest to wymiana danych pomiędzy procesami. Podstawowym mechanizmem umożliwiającym taką komunikację są łącza (pipes). Np.
who | sort | more
To połączenie trzech procesów dwoma łączami. Dane w tym przypadku przepływają w jednym kierunku.

Łącza nazwane i nienazwane
Na tym laboratorium będą omawiane i używane tzw. łącza nienazwane. Łącza nazwane to inaczej FIFO (następne laboratorium).

- Czym jest komunikacja międzyprocesowa ?
- Czym są nienazwane łącze (unnamed pipes) ?

2 pipe

Jest to funkcja, której użycie spowoduje stworzenie kanału komunikacyjnego pomiędzy dwoma procesami.

```
// pipe-create pipe
#include <unistd.h>
int pipe(
  int pfd[2] /* file descriptors */
); /* Returns 0 on success or -1 on error (sets errno) */
```

2.1. Argumenty wywołania

- int pdf[2] - tablica przechowująca deskryptory do plików przy pomocy których odbywać się będzie komunikacja. pdf[1] to plik do którego zapisujemy dane do łącza, a pdf[0], to plik z którego czytamy dane z łącza.

2.2. PIPE_BUF

Jest to stała definiująca ile danych może zapisać dany proces/wątek do łącza bez przerwania. Wartość ta jest nie mniejsza niż 512, oznacza to, że przy zapisywaniu zawsze 512 bajtów danych nie będzie przerwane przez inne dane. Wartość tą można sprawdzić przy użyciu wywołania fpathconf.

2.3. Przykład wywołania pipe ze sprawdzeniem PIPE_BUF

```
int pfd[2]; long v;
ec_neg1( pipe(pfd) )
errno = 0;
v = fpathconf(pfd[0], _PC_PIPE_BUF);
if (v == -1)
  if (errno != 0)
    EC_FAIL
  else
    printf("No limit for PIPE_BUF\n");
else
  printf("PIPE_BUF = %ld\n", v);
```

2.4. Wartość zwracana

Funkcja zwraca 0 w przypadku sukcesu oraz kod błędu w przypadku błędu.

- Do czego służy funkcja pipe ?
- Jakie są argumenty wywołania funkcji pipe ?
- Czym jest PIPE_BUF ? Jak sprawdzić wartość PIPE_BUF ?

3 write, read, close w odniesieniu do plików łączy

Niektóre wywołania systemowe zachowują się inaczej gdy argumentem jest inny plik, a inaczej w przypadku plików reprezentujących łącza. Do wywołań tych należą write, read, close. Oprócz tego działanie tych funkcji przy wywołaniach dla plików łączy jest zależne od flagi O_NONBLOCK . Flagę tę można ustawić przy pomocy funkcji fcntl.

3.1. write

write defaultowo (flaga O_NONBLOCK wyzerowana) zapisuje dane do łącza aż stanie się ono pełne, po czym zapis jest blokowany do czasu gdy jakiś inny proces odczyta część danych (read). Maksymalny rozmiar łącza to omówione już PIPE_BUF. Nie ma natomiast możliwości zapisu danych częściowych. Dane są zapisywane do łącza są w takiej kolejności jak je tam wpisano. Jeśli flaga O_NONBLOCK jest ustawiona, to dane są wpisywane natychmiast lub zwracany jest błąd w przypadku mniejszej liczby bajtów niż bufor. W przypadku rozmiaru przekraczającego bufor możliwe jest przy tej fladze ustawionej zapisywanie częściowe.
Podsumowując:

- Jeżeli zapisujemy do łącza PIPE_BUF lub mniej, to write zawsze jest atomiczną operacją (znaczy, że nie zostanie przerwane np. przez inny proces i wykona się całe)
- Jeżeli flaga O_NONBLOCK jest wyzerowana to zapisy nigdy nie są częściowe, nawet jeśli nie są atomiczne.
- Jedyny przypadek gdy częściowe zapisy się pojawiają to gdy flaga O_NONBLOCK jest ustawiona i żądana liczba bajtów jest większa od PIPE_BUF.

- Jak zachowuje się write gdy zapisywane jest PIPE_BUF lub mniej bajtów ?
- Kiedy możliwe są (jedna sytuacja) zapisy częściowe przy wywołaniu write ?

3.2. read

read defaultowo (flaga O_NONBLOCK wyzerowana) jest blokowane dopóki przynajmniej jeden bajt jest dostępny do odczytania w łączu, chyba, że zamknięto wszystkie deskryptory plików, które zapisują do łącza, wtedy read zwraca 0. Liczba bajtów do odczytania przekazywana jako parametr do read niekoniecznie będzie odczytana (odczytane zostanie tylko tyle ile jest w buforze jeśli podamy liczbę większą od tego co jest w buforze). Gdy flaga O_NONBLOCK nie jest wyzerowana to read zwróci -1 w przypadku gdy bufor będzie pusty.
Podsumowując:

- Kiedy wszystkie deskryptory plików zapisujących są zamknięte, to read czyta EOF i zwraca 0.
- Jeśli łączę jest puste, a pliki zapisujące otwarte, to read blokuje lub nie w zależności od flagi O_NONBLOCK
- Jeśli łączę nie jest puste, to read zawraca od razu i zwraca liczbę przeczytanych bajtów (niekoniecznie równą tej żądanej)
- Nie można dopuścić dwóch procesów do czytania tego samego łącza, ponieważ nie ma gwarancji atomiczności odczytu.

- Jak zachowuje się read kiedy zamknięte są deskryptory wszystkich plików zapisujących ?
- Jak zachowuje się read gdy łączę jest puste a pliki zapisujące otwarte ?
- Jak zachowuje się read w przypadku gdy łączę nie jest puste ?
- Czy można dopuścić dwa lub więcej procesów do czytania z łącza ? Dlaczego ?

3.3. close

close w przypadku łączy nie tylko zwalnia deskryptory plików dla ponownego użycia, ale również powoduje, że:

- w przypadku zamknięcia wszystkich plików zapisujących - działa tak jak EOF dla read
- w przypadku zamknięcia wszystkich plików czytających - generowany jest błąd przy zapisie do łącza

- Co się stanie z czytaniem gdy zamkniemy wszystkie pliki zapisujące ?
- Co się stanie z zapisywaniem gdy zamkniemy wszystkie pliki czytające ?

4 Komunikacja pomiędzy procesami

Komunikacja pomiędzy procesami musi być realizowana pomiędzy procesem potomnym i procesem, który jest wyżej w hierarchii (stworzył proces potomny lub jest rodzicem któregoś z jego rodziców). Dzieje się tak dlatego, że nie możemy przekazywać deskryptorów plików pomiędzy procesami, mogą być one jednak dziedziczone przez proces potomny.

- Pomiędzy jakimi procesami może być realizowana komunikacja ?

5 STDIN_FILENO i STDOUT_FILENO

Są to makra reprezentujące deskryptory odpowiadające standardowemu wejściu oraz standardowemu wyjściu. Deskryptory o numerach 0 (STDIN_FILENO) i 1 (STDOUT_FILENO) są zarezerwowane dla tych plików.

6 Funkcje dup oraz dup2

Funkcje te są odpowiedzialne za kopiowanie deskryptorów plików. Nowo powstałe deskryptory odnoszą się do tych samych plików na które wskazują deskryptory na których rzecz wywoływane są funkcje.

6.1. Funkcja dup

Funkcja ta tworzy kopię deskryptora do pliku. Argumentem wywołania tej funkcji jest deskryptor do pliku, który chcemy skopiować. WAŻNE: Funkcja ta zwraca najniższy aktualnie wolny numer deskryptora. (Pamiętamy, że 0 i 1 są automatycznie zarezerwowane przez standardowe wejście i wyjście).

```
// dup - duplicate file descriptor
#include <unistd.h>
int dup(
  int fd /* file descriptor to duplicate */
);
/* Returns new file descriptor or -1 on error (sets errno) */
```

6.2. Funkcja dup2

Funkcja ta tworzy kopię deskryptora do pliku tak jak funkcja dup, ale kopia ta jest umieszczana w istniejącym już deskryptorze. Deskryptor ten jest zamykany w razie potrzeby i nadpisywany jest tym wskazywanym przez argument fd. Istniejący deskryptor do którego ma być zapisany nowy przekazujemy jako argument fd2 do funkcji.

```
// dup2 - duplicate file descriptor
#include <unistd.h>
int dup2(
  int fd, /* file descriptor to duplicate */
  int fd2 /* file descriptor to use */
); /* Returns new file descriptor or -1 on error (sets errno) */
```

- Czym są deskryptory STDIN_FILENO i STDOUT_FILENO ? Jakie mają numery ?
- Jak działają funkcję dup i dup2 ? Czym się różnią ? Jakie mają argumenty ?