Instrukcja do laboratorium Systemów Operacyjnych

(semestr drugi)

Ćwiczenie trzecie

(jedne zajęcia)

Temat: Potoki i łącza nazwane w Linuksie.

Opracowanie:

dr inż. Arkadiusz Chrobot

Wprowadzenie

1. Komunikacja z wykorzystaniem strumieni

W systemach uniksowych istnieje możliwość utworzenia nowego procesu realizującego wybrane polecenie powłoki i stworzenia dla niego łącza komunikacyjnego za pomocą tyko jednego wywołania funkcji. Tą funkcją jest funkcja *popen()*. Łącze utworzone przy jej pomocy jest jednokierunkowe, tzn. proces wywołujący może do niego pisać lub z niego czytać, ale nigdy te dwie operacje nie są dostępne jednocześnie dla tego samego łącza. Łącze należy zamykać za pomocą funkcji *pclose()*. Jest ono wiązane ze standardowym wejściem lub wyjściem polecenia, w zależności od rodzaju operacji, czyli domyślnie „zastępuje" klawiaturę lub ekran.

2. Potoki

Inną formą komunikacji jednokierunkowej są potoki (łącza nienazwane), które zwykle służą do wymiany informacji między dwoma spokrewnionymi procesami. Choć istnieje możliwość używania potoku tylko w obrębie jednego procesu, to nie jest to rozwiązanie, które stosuje się w praktyce. Potok tworzony jest w przestrzeni jądra, ale poprzez wywołanie funkcji *pipe()* w przestrzeni użytkownika, a dostęp do niego odbywa się przy pomocy funkcji umożliwiających niskopoziomowe operacje na plikach – *read()* i *write()*. Ma on skończoną pojemność, której wielość zależy od konfiguracji systemu. W nowszych wersjach Linuksa wynosi ona 64KiB. Dla utworzonego potoku można ustawić znacznik O_NONBLOCK, który powoduje, że program nie będzie przechodził w stan oczekiwania w określonych sytuacjach, związanych z obsługą potoku (szczegóły w następnym punkcie).

3. Łącza nazwane, czyli kolejki FIFO

Kolejka jest podobna w działaniu do potoku, ale w przeciwieństwie do niego ma nazwę, a więc mogą z niej korzystać procesy niespokrewnione. Kolejki FIFO pojawiły się oficjalnie w systemach uniksowych zgodnych z wersją o nazwie System V (właściwie to istniały już od System III). Pierwotnie tworzono je za pomocą funkcji mknod(), ale obecnie istnieje wygodniejsza w użyciu funkcja mkfifo(). Aby skorzystać z tak utworzonego łącza nazwanego należy go otworzyć albo do zapisu, albo do odczytu (to samo ograniczenie co w przypadku potoku). Dodatkowo można zastosować flagę O_NONBLOCK. Powoduje to, że proces nie będzie czekał na zakończenie pewnych operacji związanych z obsługą kolejki. Dokładniej objaśnia to poniższa tabela:

Sytuacja	Nie ustawiono znacznika O_NONBLOCK	Ustawiono znacznik O_NONBLOCK
Próba otwarcia kolejki tylko do czytania, żaden proces nie otworzył jej do pisania.	Oczekiwanie, aż proces otworzy kolejkę do pisania.	Natychmiastowy powrót bez sygnalizowania błędu.
Próba otwarcia kolejki tylko do pisania, żaden proces nie otworzył jej do czytania.	Oczekiwanie, aż proces otworzy kolejkę FIFO do czytania.	Natychmiastowy powrót z sygnalizacją błędu.
Próba czytania danych z kolejki, jeśli nie ma w niej danych.	Oczekiwanie, aż pojawią się dane. Jeśli żaden proces nie otworzył kolejki do pisania, to funkcja zwraca wartość zero. Jeśli w kolejce pojawią się dane, to funkcja zwróci liczbę oczytanych bajtów.	Jeśli kolejka jest pusta to funkcja natychmiast powraca, zwracając wartość -1. Jeśli kolejka nie jest otwarta do zapisu, to funkcja również natychmiast wraca zwracając wartość 0.
Próba zapisania do zapełnionego łącza FIFO.	Oczekiwanie, aż będzie miejsce do zapisania nowych danych.	Funkcja write() wraca natychmiast zwracając wartość 0.

Dwa ostatnie wiersze powyżej tabeli odnoszą się także do potoków. Można również wyodrębnić kilka reguł, którym podlega pisanie i czytanie do kolejek i potoków:

- Jeśli proces żąda przeczytania mniejszej porcji danych niż wynosi bieżąca zawartość medium, to będzie pobrane dokładnie tyle danych, ile zażądano. Reszta danych nie ulega zniszczeniu i może być odczytana przy następnej operacji czytania.
- Jeśli proces będzie usiłował odczytać więcej danych niż znajduje się w medium, to odczytanych i tak zostanie tylko tyle danych, ile jest w potoku lub kolejce.
- Jeśli proces próbuje czytać z medium, które nie zostało przez żaden inny proces otwarte do pisania, to wynikiem funkcji read() będzie zero. W przypadku kiedy ustawiony jest znacznik O_NONBLOCK zachowanie funkcji read() jest takie samo.
- Zapis danych jest operacją niepodzielną, o ile proces zapisuje dane do medium porcjami mniejszymi od jego pojemności,

- Jeśli proces próbuje zapisywać do medium, które nie zostało otwarte przez inny proces do odczytu, to otrzyma sygnał SIGPIPE, którego domyślna obsługa polega na zakończeniu procesu.

Po skorzystaniu z kolejki należy ją usunąć, aby nie zajmowała miejsca na dysku.

4. Opis ważniejszych funkcji

- *popen()* - funkcja uruchamia polecenie powłoki podane w jej argumentach wywołania oraz tworzy strumień służący do komunikacji między procesem wywołanym a wywołującym, który można obsługiwać standardowymi funkcjami fread() i fwrite(). Szczegóły: man 3 popen.

- *fread()* - służy do odczytu buforowanego ze strumienia. Szczegóły: man 3 fread.

- *fwrite()* - służy do zapisu buforowanego do strumienia. Szczegóły: man 3 fwrite.

- Funkcja *pclose()* - służy do zamykania strumienia stworzonego przez *popen()*. Szczegóły: man 3 pclose.

- Funkcja *pipe()* - służy do tworzenia potoku łączącego dwa spokrewnione procesy. Jako argument wywołania przyjmuje dwuelementową tablicę, w której będą zapisane deskryptory potoku. Deskryptor zerowy jest do odczytu, a pierwszy do zapisu. Zwykle jest ona wywoływana przed fork(), co powoduje, że procesy powstałe w wyniku podziału odziedziczą tablicę deskryptorów. Każdy z tych procesów zamyka jeden z deskryptorów, np. jeśli komunikacja przebiega według schematu: rodzic -> potomek, to rodzic zamyka deskryptor do odczytu, a potomek do zapisu. Szczegóły: man 2 pipe

- Funkcja *read()* - czyta określoną liczbę bajtów z deskryptora bez buforowania. Szczegóły: man 2 read.

- Funkcja *write()* - zapisuje określoną liczbę bajtów do deskryptora bez buforowania. Szczegóły: man 2 write.

- Funkcja *close()* – zamyka deskryptor pliku. Do zamykania strumieni służy *fclose()* lub *pclose()*. Szczegóły: man 2 close.

- Funkcja mkfifo() - tworzy łącze nazwane o podanej nazwie i atrybutach, może być zastąpiona przez *mknod()*. Szczegóły: man 3 mkfifo

- Funkcja *open()* – otwiera plik (również łącze nazwane). Możliwe jest dzięki niej ustalenie trybu otwarcia i ustawienie znaczników. Szczegóły: man 2 open.

- Funkcja *fcntl()* - służy do manipulacji deskryptorem pliku. Można dzięki niej ustawić flagę O_NONBLOCK dla potoku. Szczegóły: man 2 fcntl.

- Funkcja *perror()* – wypisuje wiadomość o błędzie zwróconą przez system. Przykład użycia: if(fork() < 0) perror("fork"); Szczegóły: man 3 perror.

- Funkcja *unlink()* – funkcja ta usuwa z systemu plików nazwę, która może odnosić się do dowiązania lub pliku (ostatni przypadek odpowiada operacji usunięcia pliku). Można ją wykorzystać do automatycznego usuwania łącza nazwanego. Szczegóły: man 2 unlink.

Zadania:

1. Napisz program, który wywoła polecenie „sort nazwa.c", gdzie nazwa.c jest nazwą pliku z kodem źródłowym programu, a następnie odczyta 20 znaków zwróconych przez to polecenie i wyświetli je na ekran.
2. Napisz program, który uruchomi polecenie „wc" i przekaże mu na standardowe wejście dowolny ciąg znaków.
3. Napisz program, który stworzy potok, i wykorzysta go do przesyłania danych w obrębie jednego procesu.
4. Napisz program, który stworzy potok i wykorzysta go do komunikacji między dwoma spokrewnionymi procesami. Zadanie wykonaj dla dwóch przypadków: z ustawioną flagą O_NONBLOCK i bez.
5. Napisz program, który podzieli się na dwa procesy komunikujące się przy pomocy potoków. Komunikacja musi być dwukierunkowa.
6. Napisz dwa niezależne programy, które będą się komunikowały przy pomocy kolejki FIFO. Zadanie wykonaj w dwóch wariantach, tak jak zadanie 4. Po zakończeniu komunikacji kolejkę należy usunąć.
7. Napisz program, w którym komunikacja między spokrewnionymi procesami będzie odbywała się w jednym kierunku za pomocą łącza nienazwanego, a w drugim za pomocą łącza nazwanego.

8. Stwórz programy do dialogu między dwoma użytkownikami w systemie. Do komunikacji użyj łącza FIFO stworzonego za pomocą funkcji *mknod.*

9. Napisz program, który wygeneruje cztery procesy. Każdy z tych procesów będzie się komunikował z następnym za pomocą łącza nienazwanego. Pierwszy proces wyśle przez swoje łącze liczby 1,2,3 i 4. Każdy kolejny zwiększy każdą z nich o 1, a ostatni wypisze je na ekranie.

10. Napisz trzy programy komunikujące się przez łącza FIFO. Pierwszy program będzie wysyłał kolejne liczby parzyste, drugi kolejne liczby nieparzyste, a trzeci będzie odbierał te liczby i je sumował. Wszystkie procesy powinny wyświetlać wyniki pracy na ekranie.