Inter Proces Communications (IPC)- POTOKI

Rozważmy następujący potok:

```
who | cut -f1 -d  " " | sort -u | wc -l
```

Potok ten podobnie jak każdy inny służy do przeniesienia danych od jednego procesu do drugiego. W tym przykładzie jeden proces pisze na jednym końcu, potoku , drugi zaś czyta z drugiego końca potoku. W tym przypadku zapisywany koniec jednego potoku zostaje ustawiony przez powłokę jako standardowe wyjście polecenia who, a czytany koniec ustawiany jest jako standardowe wejście polecenie cut , itd.

W systemach unixowych do tworzenia potoku tkzw. pipe służy odpowiednia funkcja systemowa:

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

biblioteka: <unistd.h>

Za pomocą pipe() tworzymy kanał komunikacji międzyprocesowej (Inter Proces Communication), nazywany często pipe. Jako argument podajemy dwuelementową tablicę elementów typu int. Funkcja ustawia wartość dwóch deskryptorów plików :

- fd[0] – deskryptor służący do odczytu
- fd[1] – deskryptor służący do zapisu

Funkcja zwraca 0 w przypadku powodzenia utworzenia strumienia, -1 w przypadku błędu.

Można zatem wyobrazić sobie zastosowanie mechanizmu pipe do komunikacji jednokierunkowej między procesami:

Taki mechanizm jest najprostszym scenariuszem komunikacji, mamy tylko możliwość komunikacji jednokierunkowej, komunikację dwukierunkową można zaimplementować za pomocą jednej pipe lecz z dodatkową synchronizacją lub też za pomocą dwóch jednokierunkowych potoków. Utworzony mechanizm to kolejka FIFO.

Do przesyłania danych w potoku służą funkcje:

Biblioteka: <unistd.h>

```
int read (int fd, void *buf,size_t count)
```

Funkcja odczytuje dane dostępne pod deskryptorem pliku fd, zapisując je do załączonego bufora buf o maksymalnym rozmiarze równym count bajtów. Zwrócona wartość reprezentuje liczbę odczytanych bajtów.

```
int write (int fd, const void *buf,size_t count)
```

Funkcja zapisuje dane z bufora buf o całkowitej długości count do deskryptora pliku fd. Zwracana wartość odpowiada liczbie zapisanych danych przez funkcję . Zwykle powinna to być wartość równa count.

```
int close (int fd)
```

Funkcja powoduje zamknięcie pliku lub strumienia określodego po danym deskryptorem fd. Przy poprawnym działaniu funkcja zwraca 0.

Jeżeli któraś z powyższych trzech funkcji zwróci wartość –1 oznacza, że wystąpił błąd a jego przyczynę zawiera zmienna errno (<errno.h>).

Przykładowe zastosowanie :

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

char *mesg="DZIEN DOBRY"

int main (){
int ilosc 12;
char buf [ilosc];
int fd[2];
if (pipe(fd)==-1){
perror ("BLAD TWORZENIA POTOKU !");
}
write (fd[1],mesg,ilosc);
read (fd [0], buf, ilosc);
printf ("%s\n",buf);

exit (0);
}
```

ZADANIA:

1. Napisz program obrazujący, że rzeczywiście pipe tworze kolejkę FIFO (rozbuduj przedstawiony przykład tak, by zapisywać i odczytywać kilka informacji).
2. Dla procesów będących w relacji przodek potomek stwórz mechanizm pozwalający na stworzenie komunikacji jedno kierunkowej – pozamykaj odpowiednie deskryptory.
3. Dla procesów będących w relacji przodek potomek stwórz mechanizm pozwalający na stworzenie komunikacji dwu kierunkowej – pozamykaj odpowiednie deskryptory.
4. Korzystając z programu z zad 3 napisz program przesyłający tekst z przodka do potomka i zamieniający wszystkie litery na małe i odsyłający z powrotem do przodka.