🧡

ЛАБ 25

> **25. Связь через программный канал**
>
> Напишите программу, которая создает два подпроцесса, взаимодействующих через программный канал. Первый процесс выдает в канал текст, состоящий из символов верхнего и нижнего регистров. Второй процесс переводит все символы в верхний регистр, и выводит полученный текст на терминал. Подсказка: см. **toupper(3)**

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <ctype.h>

void main() {
        pid_t firstChild, secondChild;
        int fields[2];
        char stringToWrite[256], stringToRead[256];

        if(pipe(fields) == -1) {
                perror("Pipe error\n");
                exit(EXIT_FAILURE);
        }

        if((firstChild = fork()) == -1) {
                perror("Fork error\n");
                exit(EXIT_FAILURE);
        }

        if(firstChild == 0) {
                close(fields[0]);

                scanf("%s", stringToWrite);

                if(write(fields[1], stringToWrite, strlen(stringToWrite)) != strlen(stringToWrite)) {
                        perror("Write error\n");
                        close(fields[1]);
                        exit(EXIT_FAILURE);
                }

                close(fields[1]);
                exit(EXIT_SUCCESS);
        }

        if((secondChild = fork()) == -1) {
                perror("Fork error\n");
                exit(EXIT_FAILURE);
        }

        if(secondChild == 0) {
                close(fields[1]);

                if(read(fields[0], stringToRead, sizeof(stringToRead)== -1) {
                        perror("Read error\n");
                        close(fields[0]);
                        exit(EXIT_FAILURE);
                }

                for(size_t i = 0; i < strlen(stringToRead); i++)
                        stringToRead[i] = toupper(stringToRead[i]);

                close(fields[0]);
                printf("Got message: %s\n", stringToRead);
                exit(EXIT_SUCCESS);
        }

        close(fields[0]);
        close(fields[1]);
```

```
        while(wait(NULL) != -1) continue;
}
```

▼ **ТРУБЫ**

Часто параллельные процессы должны каким-то образом взаимодействовать для решения общей задачи, и поэтому нужны средства для обмена информацией между ними

Программные каналы(представляют собой непрерывный поток байтов**)** — это механизм передачи информации от одного процесса к другому. Линии связи между 2 или более процессами

Программные каналы, в отличие от регулярных файлов(дисковые файлы / хранилище информации, лежащие в файловой системе), представляют собой непрерывный поток байтов, по которому может быть передано произвольно большое количество информации

Трубы(= ПК)- примитивы межпроцессорных взаимодействий. Эти примитивы - файлы специального типа - fifo файлы (не регулярные, не устройства, а что-то типо псевдоустройства)

При создании создаются 2 файловых дескриптора: в один читаем, в другой пишем. В солярке можно читать и писать с двух сторон, но чтение будет перекрестным. С конца нельзя читать, если с него уже писали

Какая информация теряется при буферизации? - время прихода данных

Достоинства буферизации: потоки, которые обмениваются данными через буфер, не обязаны знать друг друга, чтобы получить буфер + повышает среднюю скорость передачи данных

Труба реализована через кольцевой буфер - он конечного размера. Если в трубу пишут, а с др стороны не читают - блокируется запись. И наоборот - блокируется чтение

Чтение - читает все, что есть в буфере. Тк чтение разрушающее ⇒ читать в несколько потоков/процессов опасно

Запись - если больше буфера - блокировка, пока все данные не пройдут через трубу. Если несколько процессов, то по очереди

lseek(), mmap() - не используются

Главное использование труб - конвейеры Shell - | - позволяет объединить 2 команды в конвейер

▼ **PIPE()**

Неименованная труба - нельзя открыть или создать при помощи open(). Создается вызовом:
int pipe (int fields[2]) - неименованная труба

Возврат:

- 0 - успех. Возвращает **2 ручки (дескрипторы файлов, использующиеся для чтения и записи. Хранятся в памяти процесса)** (1ая для чтения, 2ая для записи, но в SVR4 оба дескриптора открыты для чтения и записи, позволяя двусторонний обмен данными)
- -1 - неуспех + errno

  Тк получаем 2 ручки в 1 процессе - fork() чтобы передать открытую ручку потомку. Неименованные трубы годятся для связи родственных процессов

При **fork(2)** происходит дублирование файлового дескриптора, а программный канал считается закрытым только когда будут закрыты все копии связанного с этим каналом дескриптора.
Если вы забудете закрыть один из дескрипторов, процесс, ожидающий конец файла в канале, никогда его не дождётся.

▼ **WRITE()**

ssize_t write (int fildes , const void * buf , size_t nbyte) - Системный вызов write() пытается записать *n* байтов из буфера, на который указывает *buf* , в канал, связанный с дескриптором *fildes* .

Возврат:

- **Успех:** число байт фактически записанных в канал ≤ *nbytes*
- **Неудача:** -1 + errno

Если канал не имеет места для записи всех данных, write(2) останавливается. Система не допускает частичной записи: write(2) блокируется до момента, пока все данные не будут записаны, либо пока не будет обнаружена ошибка

- Если процесс пытается писать в канал, из которого никто не читает, он получит сигнал SIGPIPE
- Если процесс пишет и другой конец закрыли - SIGPIPE

write(2) стремится записать все данные, запись которых была запрошена.

- Если кол-во байт, которые должны быть записаны, больше свободного пространства в канале, пишущий процесс остановится, пока читающий процесс не освободит достаточно места для записи.
- Ядро обеспечивает атомарность записи: если 2+ процессов пишут в один канал, то система поставит их в очередь, так что запись следующего процесса в очереди начнётся только после того, как закончится запись предыдущего.
- Если читающий процесс закроет свой конец канала, все пишущие процессы получат сигнал SIGPIPE при попытке записи в этот канал. Это приведёт к прерыванию вызова write(2) и, если сигнал не был обработан, к завершению пишущего процесса

▼ **READ()**

ssize_t read(int fildes, void *buf, size_t nbyte) - Системный вызов read() пытается прочитать n байтов из канала, связанного с дескриптором fildes, в буфер, на который указывает buf.

Возврат:

- **Успех:** число байт фактически считанных из канала
- **Неудача:** -1 + errno

В отличие от обычных файлов, чтение разрушает данные в канале. Это означает, что вы не можете использовать lseek(2) для попыток прочитать данные заново.

Этот системный вызов читает столько данных, сколько на момент вызова есть в канале.

- Если количество байтов в канале меньше, чем требуется, read(2) возвращает значение меньшее, чем его последний аргумент.
- Возвращает 0, если обнаруживает, что другой конец канала закрыт, т.е. все потенциальные пишущие процессы закрыли свои файловые дескрипторы, связанные с входным концом канала.
- Если пишущий процесс опередил читающий, может потребоваться несколько операций чтения, прежде чем read(2) возвратит 0, показывая конец файла.
- Если буфер канала пуст, но файловый дескриптор другого конца ещё открыт, read(2) будет заблокирован. Это поведение может быть изменено флагами O_NONBLOCK и O_NDELAY.