OS Lab 25

Status	ready
checkbox	☐
class	OS
due date	@Feb 6, 2021

Task

💡 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)`

What is a pipe ??

Программные каналы - это линии связи между двумя или более процессами. По традиции, прикладные программы используют каналы следующим образом: один процесс пишет данные в канал, а другой читает их оттуда. В SVR4 каналы стали двунаправленным механизмом, так что два процесса могут передавать информацию в обоих направлениях через один программный канал.

Процессы не обязаны заботиться о переполнении канала избытком данных или о невозможности читать из пустого канала. В канальный механизм встроена синхронизация между читающим и пишущим процессами: пишущий процесс блокируется, т.е. приостанавливает исполнение, при попытке записи в переполненный канал, и, соответственно, читающий процесс останавливается при попытке чтения из пустого канала.

Канал идентифицируется таким же файловым дескриптором, как и открытые обычные и специальные файлы. Большинство системных вызовов для работы с файловыми дескрипторами применимо к каналам.

Данные пишутся в канал так же, как и в обычный файл, при помощи системного вызова `write(2)`. Как упоминалось выше, если канал не имеет места для записи всех данных, `write(2)` останавливается. Система не допускает частичной записи: `write(2)`

```
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <ctype.h>

#define TEXTLEN 30

void print_upper_string(const char* string);

int fd[2];
pid_t pid_1, pid_2;
static char textout[TEXTLEN] = "Hello, bruh!\n";
static char textin[TEXTLEN];
int status_1, status_2;

int main()
{
  if (pipe(fd) == -1) // Creating pipe
  {
    perror("Pipe ");
    exit(-1);
  }

  if ((pid_1 = fork()) == 0) // First child
  {
    write(fd[0], textout, TEXTLEN);
    exit(1);
  }
  else if (pid_1 == -1)
  {
    perror("Cannot create child ");
    exit(-1);
  }

  if (pid_1 > 0 && (pid_2 = fork()) == 0) // Second child
  {
    read(fd[1], textin, TEXTLEN);
    print_upper_string(textin);
    exit(2);
  }
  else if (pid_2 == -1)
  {
    perror("Cannot create child ");
    exit(-1);
  }

  waitpid(pid_1, &status_1, 1);
  waitpid(pid_2, &status_2, 1);

  printf("Child 1 exit status: %d\n", WEXITSTATUS(status_1));
  printf("Child 2 exit status: %d\n", WEXITSTATUS(status_2));

  return 0;
}

void print_upper_string(const char* string)
{
  char ch;
  int j = 0;

  while (string[j] != NULL)
```

блокируется до момента, пока все данные не будут записаны, либо пока не будет обнаружена ошибка.

Данные читаются из канала при помощи системного вызова `read(2)`

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

С одним каналом может работать несколько читающих и пишущих процессов. На практике, нежелательно иметь несколько читающих процессов — без дополнительной синхронизации доступа к каналу это, скорее всего, приведёт к потере данных. Но канал с несколькими пишущими в него процессами может иметь смысл.

```
  {
    ch = string[j];
    putchar(toupper(ch));
    j++;
  }
}
```

Notes

Reading list

- [ ]

Task

Notes

Reading list

- [ ]