Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Информационные технологии и прикладная математика»

# Лабораторная работа №2
# по курсу «Операционные системы»

Студент: Минеева Светлана Алексеевна

Группа: М8О-210Б-21

Преподаватель: Миронов Е.С

Вариант №14
Оценка: _____
Дата: 21.09.2022
Подпись: _____

Москва, 2022

**Содержание:**

## 1. Цель работы

Приобретение практических навыков в:

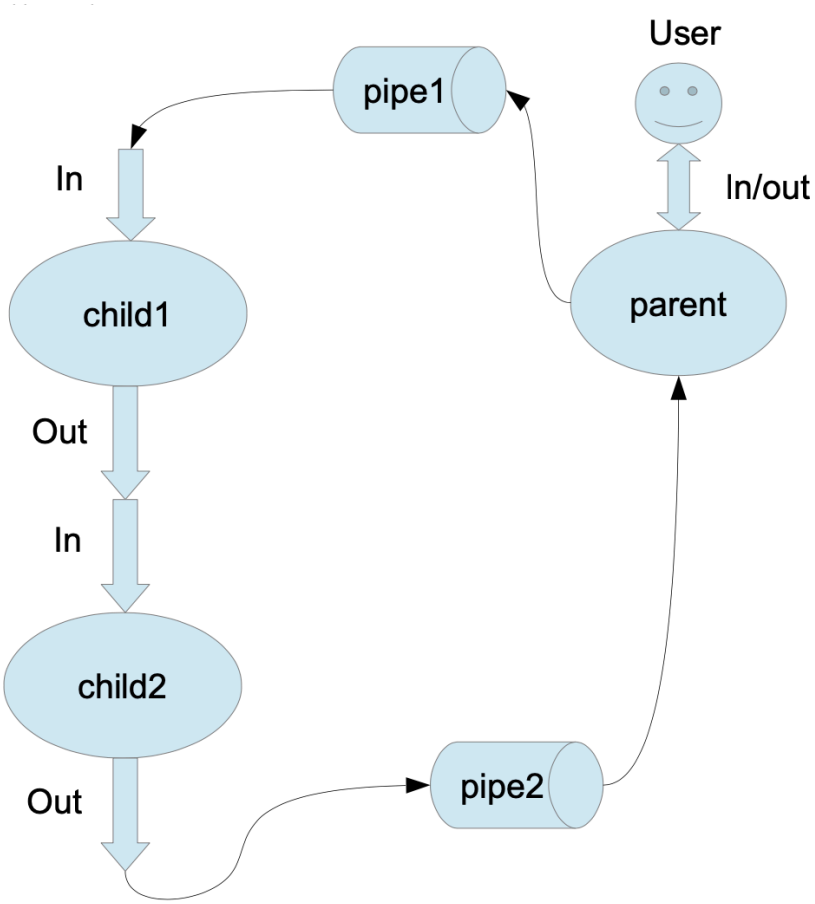
- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

## 2. Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

## 3. Вариант задания



Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Вариант №14: Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

## 4. Общие сведения о программе

Программа состоит из файлов parent.c, child1.c и child2.c. В файле parent.c храниться родительский процесс, создание дочерних процессов, перенаправление потока вывода. В child1.c строки переводятся в нижний регистр. В child2.c убираются все двойные пробелы.

Программа использует следующие системные вызовы:

1. **read –** для чтения данных из входного потока;
2. **write** – для записи данных;
3. **pipe –** для создания канала, через который процессы могут обмениваться информацией;
4. **fork –** для создания дочернего процесса;
5. **dup2** – для перенаправления потоков ввода и вывода;
6. **execve** - для замены образа памяти процесса.

## 5. Общий метод и алгоритм решения

В программе parent.c создаем дочерние процессы, в fork() проверяем создание и работу дочерних процессов. Далее создаем три канала pipe для передачи информации из родительского процесса в первый дочерний, из первого дочернего процесса во второй и из второго в родительский. Прописываем проверки для pipe. Дальше пишем в двух отдельных файлах child1.c и child2.c. выполнение задания для двух дочерних процессов: child1.c переводит строку в нижний регистр, а child2.c убирает все задвоенные пробелы. Пишем вывод программы.

# 6. Текст программы

## parent.c

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

int creation_child_processes(char *fname, int read, int write) {
  switch (fork()) {
    case -1:
      return -1;

    case 0: {
      char *args[] = {NULL};

      if (dup2(read, 0) == -1) {
        printf("dup2 error!");
      }

      if (dup2(write, 1) == -1) {
        printf("dup2 error!");
      }

      if (execv(fname, args) == -1) {
        printf("execv error!");
      }

      return 0;
    }

    default:
      break;
  }

    return 0;
}

int main() {
  int pipe1[2], pipe2[2], pipe3[2];

  if (pipe(pipe1) == -1) {
    printf("Pipe1 error!");
  }

  if (pipe(pipe2) == -1) {
    printf("Pipe2 error!");
  }

  if (pipe(pipe3) == -1) {
    printf("Pipe3 error!");
  }

  if (creation_child_processes("./child1", pipe1[0], pipe2[1])){
    perror("fork error");
    return -1;
  }

  if (creation_child_processes("./child2", pipe2[0], pipe3[1])){
    perror("fork error");
    return -1;
  }

  printf("Enter string:\n");
  char symbol;

  while ((symbol = getchar()) != EOF) {
    write(pipe1[1], &symbol, 1);
```

```
    read(pipe3[0], &symbol, 1);
    printf("%c", symbol);
    fflush(stdout);
  }

  return 0;
}
```

## child1.c

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

char to_lower(char symbol) {
  if (symbol >= 'A' && symbol <= 'Z')
    return symbol + ('a' - 'A');

  return symbol;
  }

int main() {
  char symbol;

  while ((symbol = getchar()) != EOF) {
    printf("%c", to_lower(symbol));
    fflush(stdout);
  }

  return 0;
}
```

## child2.c

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

void print_char(char symbol) {
  printf("%c", symbol);
  fflush(stdout);
}

int main() {
  char symbol;

  while ((symbol = getchar()) != EOF) {
    print_char(symbol);

    if (symbol == ' ') {
      while ((symbol = getchar()) == ' ')
        print_char(0);
      print_char(symbol);
    }
  }

    return 0;
}
```

## 7. Демонстрация работы программы

Last login: Wed Sep 21 19:54:04 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-MacBook:~ macbookpro$ ./parent
*Enter string:*
*TT  JJ*
*tt jj*
*KnH Bhy     HYTU  jkuhJ*
*knh bhy hytu jkuhj*
*KIL  KK  mRefhgR  HGUyg B*
*kil kk mrefhgr hguyg b*
*hh   nn*
*hh nn*
*56 %6Gt4*
*56 %6gt4*
*HTYUF  ghjhg*
*htyuf ghjhg*
*JJ  JJ  JJ  JJ  JJJ j*
*jj jj jj jj jjj j*
MacBook-Pro-MacBook:~ macbookpro$

## 8. Вывод

Я составила и отладила программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними. В процессе выполнения данной лабораторной работы я приобрела практические навыки в управлении ОС и обеспечении обмена данных между процессами посредством каналов. Мною были освоены разнообразные функции для работы с процессами, такие как fork(), pipe(), dup2(), execve(). Для себя я отметила, что, работая с процессами, важно помнить о следующих вещах: при использовании fork() фактически создаётся копия текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям; у каждого процесса есть свой id, по которому его можно определить; при чтении и записи из канала, надо помнить, что read() и write() возвращают количество успешно считанных/записанных байт; важно закрывать pipe после завершения работы. Также я узнала о том, как важно писать чистый, красивый код, давать переменным самодокументированные имена, ознакомилась со стандартом Google Code Style.