Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**ПРОЦЕССЫ**

Студент: Комбаров Владислав Александрович

Группа: М8О–212Б–22

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управлении процессами в ОС
* Обеспечении обмена данных между процессами посредством каналов

## 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 решения 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.

**Общие сведения о программе**

Конечным результатом выполнения задания является CMake проект, состоящий из файлов parent.c (целевой объект для запуска), child1.c, child2.c и CMakeLists.txt. Используются заголовочные файлы: stdio.h, stdlib.h, unistd.h, sys/types.h, ctype.h, string.h. В программе используются следующие системные вызовы:

1. **fork** – создание дочернего процесса.
2. **execl** – выполнение другой программы изнутри текущей программы (передаются аргументы функции)
3. **pipe** – создание канала для передачи данных между процессами (один процесс может читать данные, а другой – записывать).
4. **exit** –завершение выполнения процесса и возвращение статуса.
5. **dup2** – переназначение файлового дескриптора.
6. **close** – закрыть дескриптор.
7. **read** – чтение данных.
8. **write** – запись данных.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, exec, pipe, dup2.
2. Написать файлы дочерних процессов child1.c, child2.c для обработки строк.
3. Написать файл parent.c, в котором будут осуществляться запуски программ child1 и child2 и обмен данными между процессами.

**Основные файлы программы**

**child1.c:**

#include <stdio.h>

#include <unistd.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#define BUFFER\_SIZE 1024

int main() {

char \*buffer = malloc(BUFFER\_SIZE);

ssize\_t bytesRead;

while ((bytesRead = read(STDIN\_FILENO, buffer, BUFFER\_SIZE)) > 0) {

// Проверяем, была ли введена пустая строка

if (bytesRead == 1 && buffer[0] == '\n') {

break;

}

// Удаляем символ новой строки

if (buffer[bytesRead - 1] == '\n') {

buffer[bytesRead - 1] = '\0';

}

for (int i = 0; i < bytesRead - 1; i++) {

buffer[i] = toupper(buffer[i]);

}

// Вывод строки

write(STDOUT\_FILENO, buffer, strlen(buffer));

write(STDOUT\_FILENO, "\n", 1);

}

free(buffer);

return 0;

}

**child2.c:**

#include <stdio.h>

#include <unistd.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#define BUFFER\_SIZE 1024

int main() {

char \*buffer = malloc(BUFFER\_SIZE);

ssize\_t bytesRead;

while ((bytesRead = read(STDIN\_FILENO, buffer, BUFFER\_SIZE)) > 0) {

// Проверяем, была ли введена пустая строка

if (bytesRead == 1 && buffer[0] == '\n') {

break;

}

// Удаляем символ новой строки

if (buffer[bytesRead - 1] == '\n') {

buffer[bytesRead - 1] = '\0';

}

for (int i = 0; i < bytesRead - 1; i++) {

if (buffer[i] == ' ') buffer[i] = '\_';

}

// Вывод строки

write(STDOUT\_FILENO, buffer, strlen(buffer));

write(STDOUT\_FILENO, "\n", 1);

}

free(buffer);

return 0;

}

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#define BUFFER\_SIZE 1024

int main() {

int parent\_pid = getpid(), cpid1, cpid2;

int pipe1fd[2]; // parent to child 1

if (pipe(pipe1fd) == -1){

write(STDERR\_FILENO, "Ошибка в создании канала\n", 25 \* sizeof(char));

}

int pipefd[2]; // child 1 to child 2

if (pipe(pipefd) == -1){

write(STDERR\_FILENO, "Ошибка в создании канала\n", 25 \* sizeof(char));

}

int pipe2fd[2]; // child 2 to parent

if (pipe(pipe2fd) == -1){

write(STDERR\_FILENO, "Ошибка в создании канала\n", 25 \* sizeof(char));

}

char \*buffer = malloc(BUFFER\_SIZE);

size\_t bytesRead;

if ((bytesRead = read(STDIN\_FILENO, buffer, BUFFER\_SIZE)) > 0){

// Проверяем, была ли введена пустая строка

if (bytesRead == 1 && buffer[0] == '\n') {

return 0;

}

// Удаляем символ новой строки

if (buffer[bytesRead - 1] == '\n') {

buffer[bytesRead - 1] = '\0';

}

cpid1 = fork();

if (cpid1 == -1) {

perror("Ошибка при создании 1 дочернего процесса");

exit(EXIT\_FAILURE);

}

cpid2 = fork();

if (cpid2 == -1) {

perror("Ошибка при создании 2 дочернего процесса");

exit(EXIT\_FAILURE);

}

// Если это родительский процесс

if (cpid1 > 0 && cpid2 > 0){

close(pipe1fd[0]);

write(pipe1fd[1], buffer, BUFFER\_SIZE);

close(pipe1fd[1]);

// записываем из родительского процесса в первый дочерний

char result[BUFFER\_SIZE] = "";

close(pipe2fd[1]);

read(pipe2fd[0], result, BUFFER\_SIZE);

write(STDOUT\_FILENO, result, BUFFER\_SIZE);

write(STDOUT\_FILENO, "\n", 1);

close(pipe2fd[0]);

}

// первый дочерний процесс

else if (cpid1 == 0){

close(pipe1fd[1]);

dup2(pipe1fd[0], STDIN\_FILENO);

close(pipefd[0]);

dup2(pipefd[1], STDOUT\_FILENO);

execl("./child1", "./child1", NULL);

}

// второй дочерний процесс

else if (cpid2 == 0){

close(pipefd[1]);

dup2(pipefd[0], STDIN\_FILENO);

close(pipe2fd[0]);

dup2(pipe2fd[1], STDOUT\_FILENO);

execl("./child2", "./child2", NULL);

}

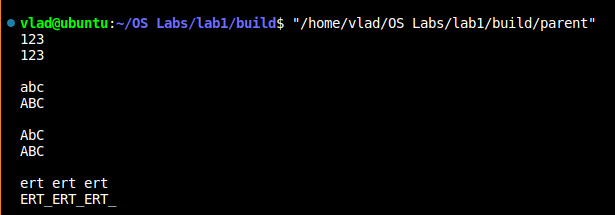
}

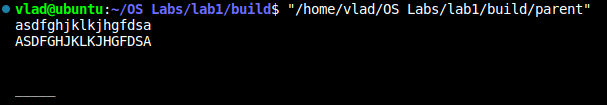
free(buffer);

return(main());

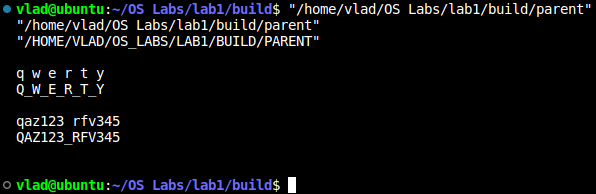
}

**Пример работы**

****

****

*Примечание:* на втором скриншоте в последнем тесте было введено пять пробелов подряд, что дало пять нижних подчёркиваний



**Вывод**

В ходе работы я приобрел навыки, необходимые для работы с процессами в операционной системе UNIX.