Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**Организация взаимодействия процессов через pipe и FIFO в UNIX**

Лабораторная работа №5 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_ Шляханов Д. А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** познакомиться с механизмами работы pipe и FIFO.

**Задание:**

Основной процесс должен сгенерировать N случайных чисел и вывести их на экран. Число N должно быть получено из параметров командной строки.

Затем требуется породить новый процесс, который выполнит сортировку.

Передать числа в него нужно через FIFO.

Второй процесс должен отсортировать числа по убыванию.

Первый процесс в это время должен ожидать сообщения от второго процесса о готовности к передаче в обратном направлении результата работы. Сообщение можно передать через уже установленный pipe или FIFO, можно создать новый pipe или FIFO, или воспользоваться любым другим способом.

Передача отсортированного массива в первый процесс должна быть организована через pipe.

Второй процесс не должен осуществлять никакого вывода на экран. После его завершения первый процесс должен вывести на экран отсортированный массив.

Первый процесс должен самостоятельно выполнить освобождение всех выделенных ресурсов перед завершением своей работы.

В отчете привести исходный код программы, а также результаты, выведенные на экран.

**Содержание файла fifo\_and\_pipe.c:**

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <fcntl.h>

const char \*NAME = "test";

int compare(const void \*a, const void \*b)

{

return(\*(int \*)b) - \*((int \*)a);

}

int \*random\_nums(int n)

{

srand((unsigned) (time(0)));

int \*nums = malloc( sizeof( int) \* n);

for( int i = 0; i < n; i++)

{

nums[i] = rand() % 1000;

}

return nums;

}

void out\_nums(int \*nums, int c)

{

for( int i = 0; i < c; i++)

{

printf("%d ", nums[i]);

}

printf("\n");

}

int main(int argv, char \*argc[])

{

int n = atoi(argc[1]);

int \*nums = random\_nums(n);

out\_nums(nums, n);

mknod(NAME, S\_IFIFO | 0666, 0);

int p[2];

pipe(p);

int child\_id = fork();

if (child\_id == 0)

{

close(p[1]);

int fifo = open(NAME, O\_WRONLY);

int \*received\_nums = malloc(sizeof(int) \* n);

read(p[0], received\_nums, sizeof(int) \* n);

close(p[0]);

qsort(received\_nums, n, sizeof(int), compare);

write(fifo, received\_nums, sizeof(int) \* n);

}

else

{

close(p[0]);

int fifo = open(NAME, O\_RDONLY);

write(p[1], nums, sizeof(int) \* n);

close(p[1]);

int \*sorted\_nums = malloc(sizeof(int) \* n);

read(fifo, sorted\_nums, sizeof(int) \* n);

out\_nums(sorted\_nums, n);

unlink(NAME);

}

}

**Результат, выведенный на экран:**

den@gremlin:~/labs/5/fifo\_and\_pipe$ make

gcc fifo\_and\_pipe.c -o fifo\_and\_pipe

./fifo\_and\_pipe 10

423 85 665 139 959 816 534 646 990 393

85 139 393 423 534 646 665 816 959 990

**Вывод:** В процессе выполнения лабораторной работы я на практике познакомился с работой pipe и FIFO в UNIX.