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.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 9091:

\_\_\_\_\_\_\_ Виноградов Ю.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 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 \*FIFO\_NAME = “novsu.fifo”;

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

{

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

}

int \*get\_random\_nums(int n)

{

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

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

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

{

nums[i] = rand() % 100000;

}

return nums;

}

void print\_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 = get\_random\_nums(n);

print\_nums(nums, n);

mknod(FIFO\_NAME, S\_IFIFO | 0666, 0);

int p[2];

pipe(p);

int child\_id = fork();

if (child\_id == 0)

{

close(p[0]);

int fifo = open(FIFO\_NAME, O\_RDONLY);

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

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

close(fifo);

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

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

close(p[1]);

}

else {

close(p[1]);

int fifo = open(FIFO\_NAME, O\_WRONLY);

write(fifo, nums, sizeof(int) \* n);

close(fifo);

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

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

print\_nums(sorted\_nums, n);

close(p[0]);

unlink(FIFO\_NAME);

}

}

**Результат, выведенный на экран:**

Werwolf2085@MacBook-Pro-werwolf2085 5 % make

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

./fifo\_and\_pipe 10

21276 63063 43492 51347 34647 66441 61617 19849 94499 45678

94499 66441 63063 61617 51347 45678 43492 34647 21276 19849

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