Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования "Новгородский государственный университет имени Ярослава Мудрого"

Кафедра «Информационных технологий и систем»

Дисциплина «Операционные системы»

Отчет по лабораторной работе

«Организация взаимодействия процессов через pipe и FIFO в UNIX»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Алентьев Александр Григорьевич/

Подпись ФИО

Принял преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_/Ананьев Владислав Валерьевич/

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работ:**

Цель работы: познакомиться с механизмом взаимодействия процессов через PIPE и FIFO в операционной системе UNIX

**Задание:**

1. Основной процесс должен сгенерировать N случайных числе и вывести их на экран. Число N должно быть получено из аргументов командной строки.
2. Требуется породить новый процесс, который выполнит сортировку.
3. Передать числа в новый процесс нужно через FIFO.
4. Второй процесс должен отсортировать числа по убыванию.
5. Передача отсортированного массива обратно в главный процесс осуществить через PIPE.
6. Главный процесс должен вывести отсортированный массив на экран.

**Ход работы:**

**Исходный текст программы:**

|  |
| --- |
| main.cpp |
| #include "stdio.h"  #include "stdlib.h"  #include "unistd.h"  #include "time.h"  #include "sys/wait.h"  #include "sys/stat.h"  #include "fcntl.h"  void print\_array\_int(int\* arr, int size)  {  for (int i = 0; i < size; i++)  printf("%d ", arr[i]);  printf("\n");  }  int compare\_int\_value(const void\* a, const void\* b)  {  return \*((int\*) b) - \*((int\*) a);  }  int main(int argv, char\* argc[])  {  if (argv <= 1)  {  printf("Error! Not enough params!\n");  return -1;  }  int arrSize = atoi(argc[1]);  int\* arr = malloc(sizeof(int) \* arrSize);  srand(time(NULL));  for (int i = 0; i < arrSize; i++)  {  arr[i] = rand() % 100;  printf("%d ", arr[i]);  }  printf("\n");  int fdPipe[2], fdFifo;  size\_t size;  if (pipe(fdPipe) < 0)  {  printf("Error! Can't create pipe!\n");  return -1;  }  const char\* name = "arr.fifo";  (void) umask(0);  if (mknod(name, S\_IFIFO | 0666, 0) < 0)  {  printf("Error! Can't create FIFO!\n");  return -1;  }  pid\_t childProcess = fork();  if(childProcess < 0) // error create process  {  printf("Error! Can't fork child!\n");  return -1;  }  else if(childProcess > 0) // process dad  {  close(fdPipe[1]);    if((fdFifo = open(name, O\_WRONLY))<0)  {  printf("Can`t open FFO for writing\n");  exit(-1);  }  size = write(fdFifo, arr, sizeof(int) \* arrSize);  if(size < (sizeof(int) \* arrSize))  {  printf("Can't write all array to FIFO!\n");  return -1;  }  close(fdFifo);  waitpid(childProcess, NULL, 0);  size = read(fdPipe[0], arr, sizeof(int) \* arrSize);  if(size < 0)  {  printf("Can't read array!\n");  return -1;  }  close(fdPipe[0]);  print\_array\_int(arr, arrSize);  }  else // process child  {  close(fdPipe[0]);  if((fdFifo = open(name, O\_RDONLY))<0)  {  printf("Can`t open F for writing\n");  exit(-1);  }  size = read(fdFifo, arr, sizeof(int) \* arrSize);  if(size < sizeof(int) \* arrSize)  {  printf("Can`t read array");  exit(-1);  }  close(fdFifo);  qsort(arr, arrSize, sizeof(int), compare\_int\_value);  size = write(fdPipe[1], arr, sizeof(int) \* arrSize);  if(size < sizeof(int) \* arrSize)  {  printf("Can`t write arr in pipe");  exit(-1);  }  close(fdPipe[1]);  }  char delete\_fifo\_file[124];  sprintf(delete\_fifo\_file, "rm %s", name);  system(delete\_fifo\_file);    free(arr);  } |

|  |
| --- |
| Output Terminal |
| >>make  gcc main.c -o main  ./main 20  12 73 24 15 82 33 67 70 1 35 84 24 93 73 19 57 68 89 26 19  93 89 84 82 73 73 70 68 67 57 35 33 26 24 24 19 19 15 12 1 |

**Вывод:**

Входе выполнения лабораторной работы, я познакомился с механизмом организации взаимодействия процессов через PIPE и FIFO в операционной системе UNIX.