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

\_\_\_\_\_\_\_\_\_\_\_\_\_/Лопатин Даниил Михайлович/

Подпись ФИО

Принял преподаватель

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

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работы**

Цель работы: познакомиться с механизмами организации взаимодействия процессов через pipe и FIFO в UNIX.

**Исходный текст программы**

|  |
| --- |
| main.c |
| #include "stdio.h"  #include "stdlib.h"  #include "unistd.h"  #include "time.h"  #include "sys/wait.h"  #include "sys/stat.h"  #include "fcntl.h"  #include <sys/cdefs.h>  void print\_arr\_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 arr\_size = atoi(argc[1]);  int\* array = malloc(sizeof(int) \* arr\_size);  srand(time(NULL));  for (int i = 0; i < arr\_size; i++)  {  array[i] = rand() % 100;  printf("%d ", array[i]);  }  printf("\n");  int FDpipe[2], FDfifo;  size\_t size;  if (pipe(FDpipe) < 0)  {  printf("Error with creating the pipe!\n");  return -1;  }  const char\* file\_name = "tmp.fifo";  (void) umask(0);  if (mknod(file\_name, S\_IFIFO | 0666, 0) < 0)  {  printf("Error with creating the FIFO!\n");  return -1;  }  pid\_t child\_process = fork();  if (child\_process == -1)  {  printf("Error with forking the child!\n");  return -1;  }  else if (child\_process == 1)  {  close(FDpipe[0]);  size = write(FDpipe[1], array, sizeof(int) \* arr\_size);  if (size != sizeof(int) \* arr\_size)  {  printf("Error! Parent can't write all array!\n");  return -1;  }  close(FDpipe[1]);  if ((FDfifo = open(file\_name, O\_RDONLY)) < 0)  {  printf("Error! Parent can't open FIFO for reading!\n");  return -1;  }  size = read(FDfifo, array, sizeof(int) \* arr\_size);  if (size < 0)  {  printf("Error! Parent can't read array using FIFO!\n");  return -1;  }  printf("Parent received this array: ");  print\_arr\_int(array, arr\_size);    close(FDfifo);  }  else if (child\_process == 0)  {  close(FDpipe[1]);  int\* new\_array = malloc(sizeof(int) \* arr\_size);  size = read(FDpipe[0], new\_array, sizeof(int) \* arr\_size);  if (size < 0)  {  printf("Error! Child can't read array!\n");  return -1;  }  qsort(new\_array, arr\_size, sizeof(int), compare\_int\_value);  close(FDpipe[0]);  if ((FDfifo = open(file\_name, O\_WRONLY)) < 0)  {  printf("Error! Child can't open FIFO for writing!\n");  return -1;  }  size = write(FDfifo, new\_array, sizeof(int) \* arr\_size);  if (size != sizeof(int) \* arr\_size)  {  printf("Error! Child can't write all array using FIFO!\n");  return -1;  }  close(FDfifo);  free(new\_array);  printf("Child process done!\n");  return 0;  }  char delete\_fifo\_file[124];  sprintf(delete\_fifo\_file, "rm %s", file\_name);  system(delete\_fifo\_file);  free(array);  return 0;  } |

**Результат выполнения программы**

|  |
| --- |
| Вывод в терминале |
| >> make  gcc main.c -o main  ./main 20  6 51 13 6 9 76 59 36 12 57 85 14 9 68 17 69 50 11 7 12  Child process done!  Parent received this array: 85 76 69 68 59 57 51 50 36 17 14 13 12 12 11 9 9 7 6 6 |

**Вывод**

Вывод: выполняя лабораторную работу, я познакомился с механизмами организации взаимодействия процессов через pipe и FIFO в UNIX.