Лабораторная работа №2

студента группы ИТ-222

Мокрищева Николая Павловича

Выполнение:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Защита: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

**UNIX**

Цель работы: ознакомление с организацией взаимодействий процессов через pipe и FIFO в UNIX.

**Содержание работы**

Вариант №7

1. Текст программы 05-1.с для записи информации в файл. Копия экрана, подтверждающего правильность выполнения программы.

2. Текст программы для чтения информации из файла. Копия экрана, подтверждающего правильность выполнения программы.

3. Текст программы 05-2.с, иллюстрирующей работу с pip'ом в рамках одного процесса. Копия экрана, подтверждающего правильность выполнения программы.

4. Текст программы 05-3.с, осуществляющей однонаправленную связь через pipe между процессом-родителем и процессом-ребенком. Копия экрана, подтверждающего правильность выполнения программы.

5. Текст программы для связи через pipe между собой двух родственных процессов, исполняющих разные программы. Копия экрана, подтверждающего правильность выполнения программы.

6. Текст программы, осуществляющей двунаправленную связь через pipe между процессом-родителем и процессом-ребенком. Копия экрана, подтверждающего правильность выполнения программы.

7. Текст программы для определения размера pipe для операционной системы. Копия экрана, подтверждающего правильность выполнения программы.

8. Текст программа 05-4.с, осуществляющей однонаправленную связь через FIFO между процессом-родителем и процессом-ребенком. Копия экрана, подтверждающего правильность выполнения программы.

9. Два текста программ, одна из которых пишет информацию в FIFO, а вторая – читает из него, так чтобы между ними не было ярко выраженных родственных связей. Копия экрана, подтверждающего правильность выполнения программы.

10. Вывод.

**Ход работы**

Вариант №7

1. Заносим информацию в файл и считываем информацию из него, проверяя корректность всех действий (Рисунок 1, 2).



Рисунок 1. Текст программы 05-1.с



Рисунок 2. Проверка файла после программы 05-1.с

2. Чтобы считать информацию из файла, мы открываем его для чтения, после чего считываем информацию, проверяем корректность и закрываем канал (Рисунок 3, 4).

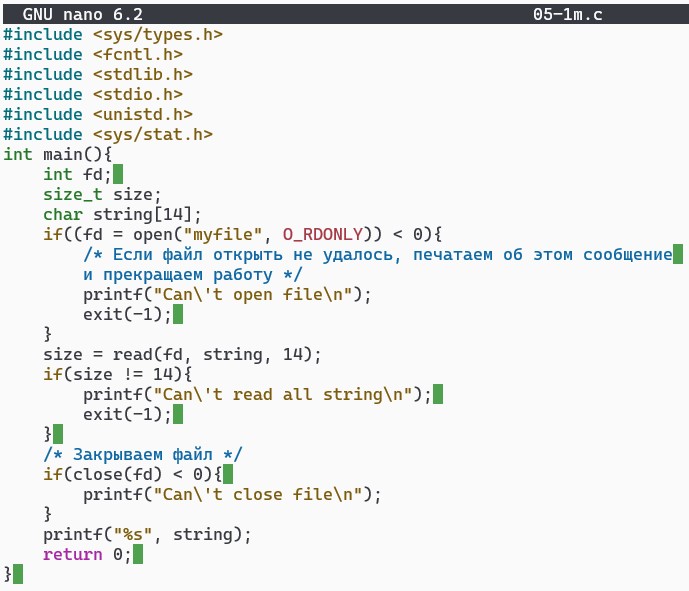


Рисунок 3. Текст программы, считывающей информацию с файла



Рисунок 4. Результат работы программы, которая считывает информацию с файла

3. Создаём pipe, чтобы записать в него строку, после чего считываем нужную информацию, выводим её и закрываем каналы (Рисунок 5, 6).



Рисунок 5. Текст программы 05-2.с



Рисунок 6. Вывод программы 05-2.с

4. Проделываем то же самое, что было в программе 05-2.с, только в этот раз запись и чтение c pipe будем производить в родительском и дочернем процессе соответственно (Рисунок 7, 8).



Рисунок 7. Текст программы 05-3.с



Рисунок 8. Вывод программы 05-3.с

5. Создаём 2 дочерних процесса, которые будут отвечать за запись в pipe и чтение из него, процесс 1 записывает строку, после чего процесс 2 получает информацию через pipe и выводит её (Рисунок 9,10, 11).



Рисунок 9. Начало текста программы для связи через pipe родственных процессов

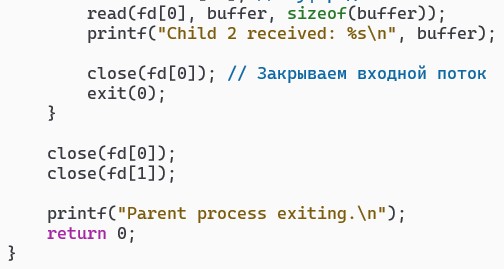


Рисунок 10. Конец текста программы для связи через pipe родственных процессов



Рисунок 11. Вывод программы для связи через pipe родственных процессов

6. Чтобы сделать двунаправленную связь, нужно создать два pipe, чтобы через один процесс 1 передавал информацию процессу 2, а через второй процесс 2 передавал информацию процессу 1. Чтобы процессы не блокировались, нужно, чтобы никакой процесс не пытался считать информацию из пустого pipe в то время, как второй не собирается ничего туда записывать (Рисунок 12, 13, 14).



Рисунок 12. Начало текста программы для создания двунаправленной связи

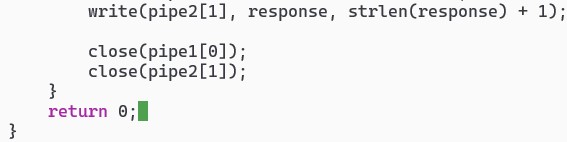


Рисунок 13. Конец текста программы для создания двунаправленной связи

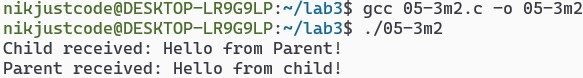


Рисунок 14. Результат работы программы для создания двунаправленной связи

7. Чтобы определить размер pipe в операционной системе можно записывать в буфер по 1 биту до того момента, как он не заполнится, чтобы при этом посчитать кол-во записанных бит, нужно сделать счётчик, который выведет кол-во записанных бит в конце программы. Чтобы процесс не заморозился в конце, были установлены флаги для канала записи, которые запрещают процессу блокироваться при работе с каналом (Рисунок 15, 16).

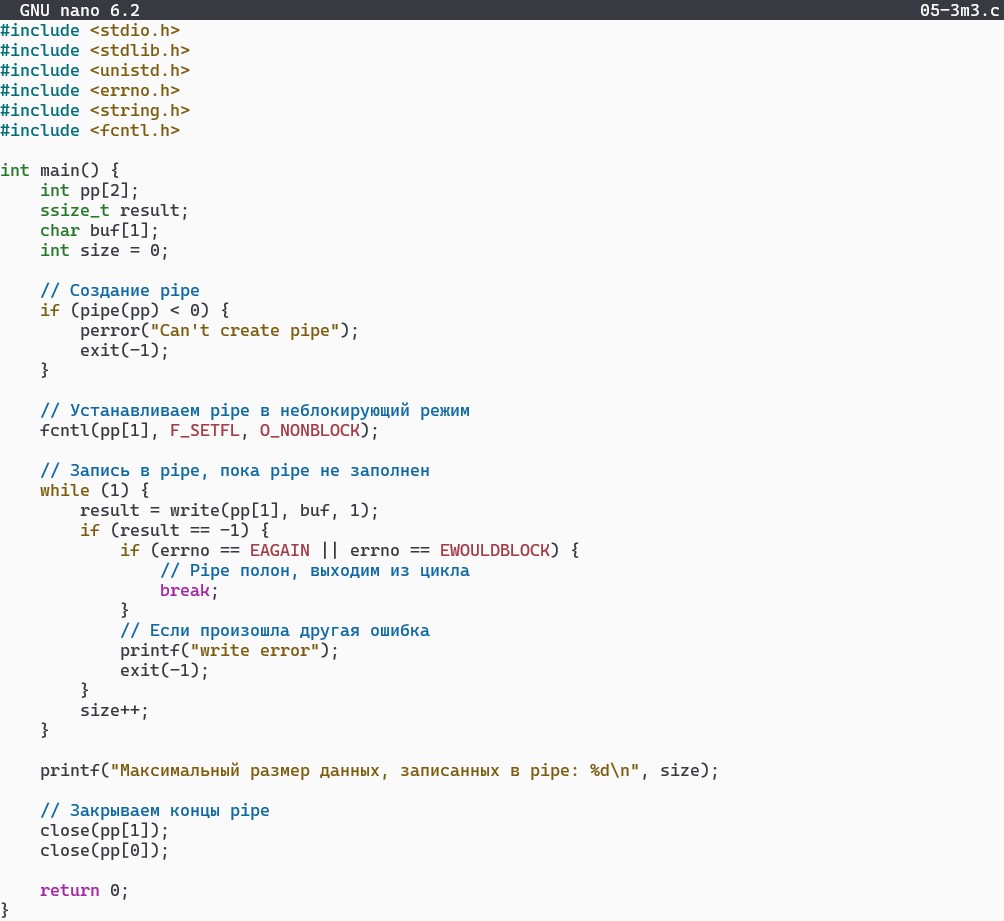


Рисунок 15. Код программы, считающей кол-во бит в pipe



Рисунок 16. Результат работы программы по подсчёту кол-ва бит в pipe

8. Чтобы работать с FIFO нужно создать fifo-файл, после чего родительский процесс передаёт информацию через канал записи, а дочерний процесс в свою очередь будет получать информацию от предка (Рисунок 17, 18, 19).

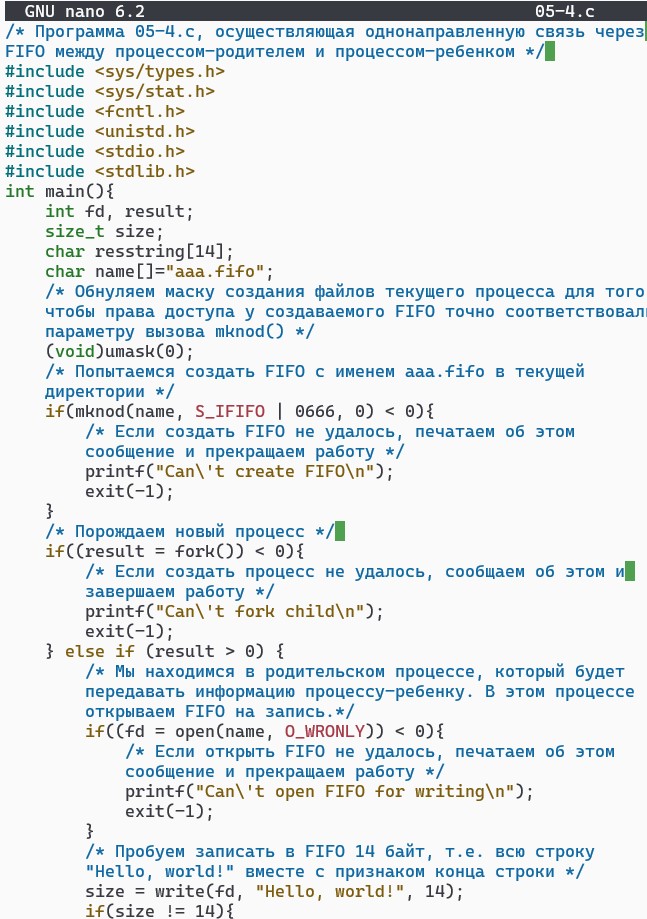


Рисунок 17. Начало текста программы 05-4.с

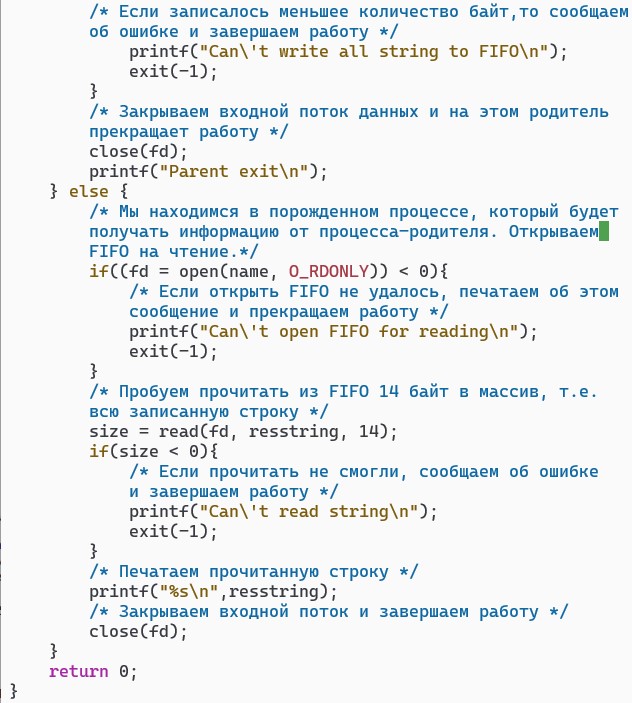


Рисунок 18. Конец текста программы 05-4.с

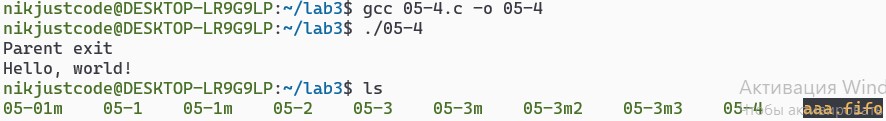


Рисунок 19. Результат работы программы 05-4.с

9. Чтобы создать два процесса, которые не будут иметь родственных связей, нужно написать 2 разные программы, которые будут взаимодействовать с файлом FIFO, которые будут записывать и считывать информацию через каналы связи, для этого нужно, чтобы изначально оба процесса знали через какой fifo-файл они будут общаться. Чтобы не случилась блокировки программы, нужно, чтобы эти программы запустились одновременно, так как процесс, который будет передавать информацию будет заморожен, пока кто-то не откроет канал для чтения, а процесс, который принимает информацию будет бесконечно ожидать, пока кто-то захочет ему что-то 'рассказать' (Рисунок 20, 21, 22).

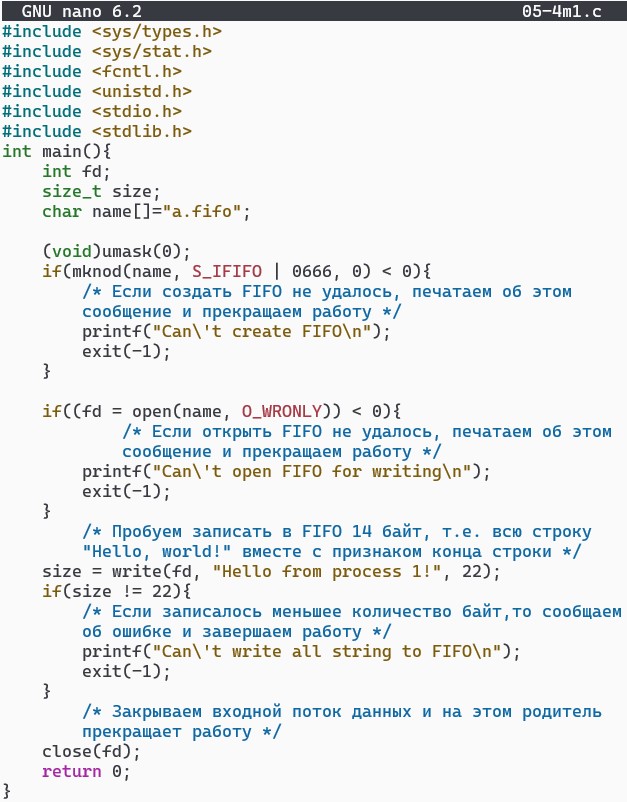


Рисунок 20. Код программы, которая будет передавать информацию для процесса-получателя

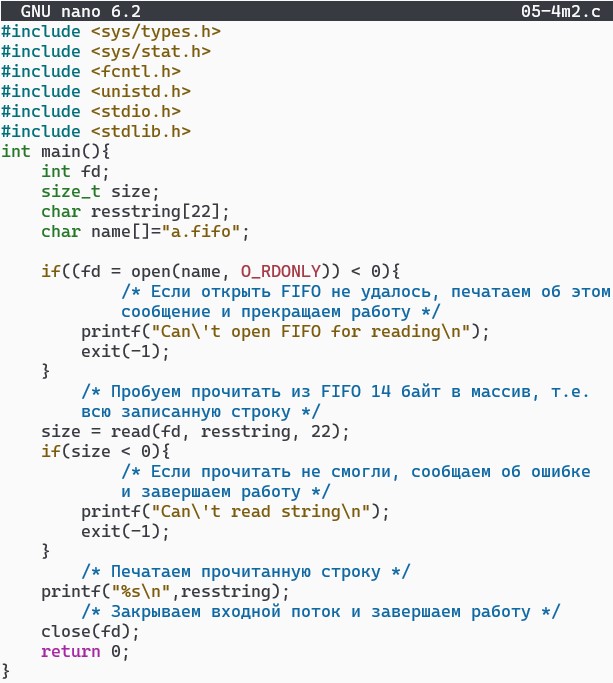


Рисунок 21. Код программы, которая будет получать информацию от процесса-отправителя



Рисунок 22. Результат работы программ после общения

10. В данной лабораторной работе мы изучили механизмы pipe и FIFO для взаимодействия процессов в UNIX, что позволяет эффективно обмениваться данными между ними.

Потоки ввода-вывода позволяют программам взаимодействовать с устройствами ввода и вывода, а файловые дескрипторы служат для доступа к открытым файлам через системные вызовы, такие как open(), close(), read() и write().

Pipe организует одностороннюю связь между процессами. Он создается с помощью pipe(), и дочерний процесс, созданный через fork(), наследует его файловые дескрипторы, что позволяет передавать данные, соблюдая блокировку операций чтения и записи.

FIFO (First In, First Out) — это специальный тип файла для обмена данными между процессами в порядке записи. FIFO создается с помощью mkfifo() или mknod(), и данные записываются и читаются через open() и write()/ read().

Таким образом, взаимодействие процессов через pipe и FIFO становится основой для разработки многопроцессных приложений. Изученные механизмы могут быть использованы в различных областях, включая сетевые приложения и обработку данных, что требовало бы дальнейшего изучения других методов межпроцессного взаимодействия, таких как сокеты и разделяемая память.