Лабораторная работа №5

студента группы ИТ-222

Мокрищева Николая Павловича

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

**Семафоры в UNIX как средство синхронизации процессов**

Цель работы: ознакомление с организацией работы с семафорами в Unix для организации синхронизации процессов.

**Содержание работы**

Вариант №7

1. Тексты программы 08-1a.c и 08-1b.c для иллюстрации работы с семафорами. Копия экрана, подтверждающего правильность выполнения программ.

2. Тексты программ 08-1a.c и 08-1b.c, измененных так, чтобы первая программа могла работать без блокировки после не менее 5 запусков второй

программы. Копия экрана, подтверждающего правильность выполнения программ.

3. Результаты выполнения в терминале команд ipcs и ipcrm при удалении семафоров.

4. Тексты модифицированных программ 06-3a.с и 06-3b.c, в которых обеспечены с помощью семафоров взаимоисключения для их правильной работы. Копия экрана, подтверждающего правильность выполнения программ.

5. Текст программы, использующей семафоры для синхронизации двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe. Копия экрана, подтверждающего правильность выполнения программ.

6. Вывод.

**Ход работы**

Вариант №7

1. Программы 08-1а.с и 08-1b.c создают и используют семафор таким образом, чтобы программа 08-1а.с могла запуститься только после хотя бы одного запуска программы 08-1b.c, так как при каждом запуске второй программы значение семафора увеличивается на 1, а при запуске первой программы уменьшалось на 1, то кол-во запусков без блокировки первого процесса была равна кол-ву запусков второго процесса. (Рисунок 1, 2, 3, 4, 5)

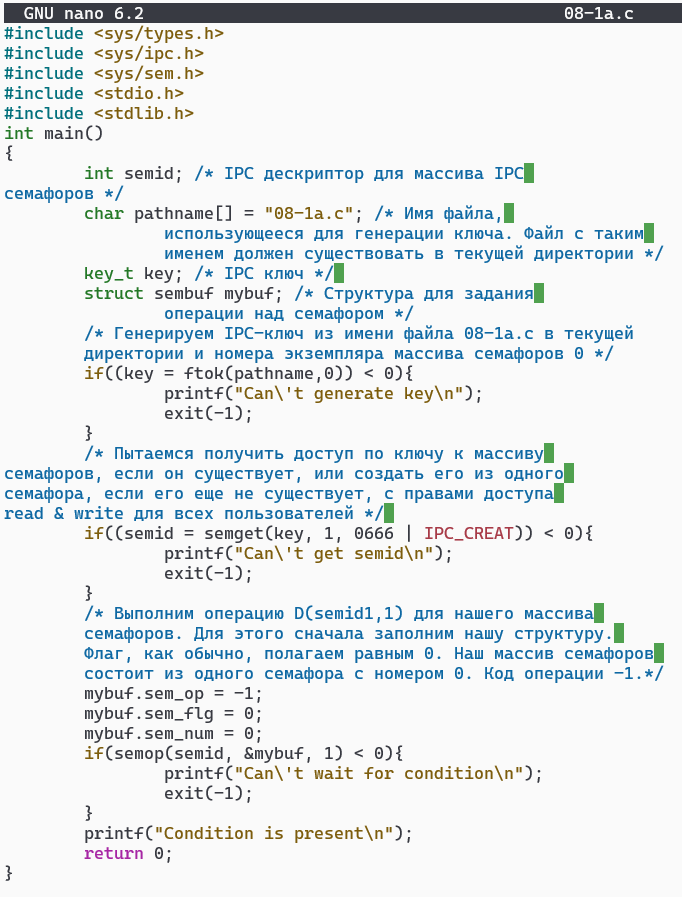


Рисунок 1. Текст программы 08-1а.с

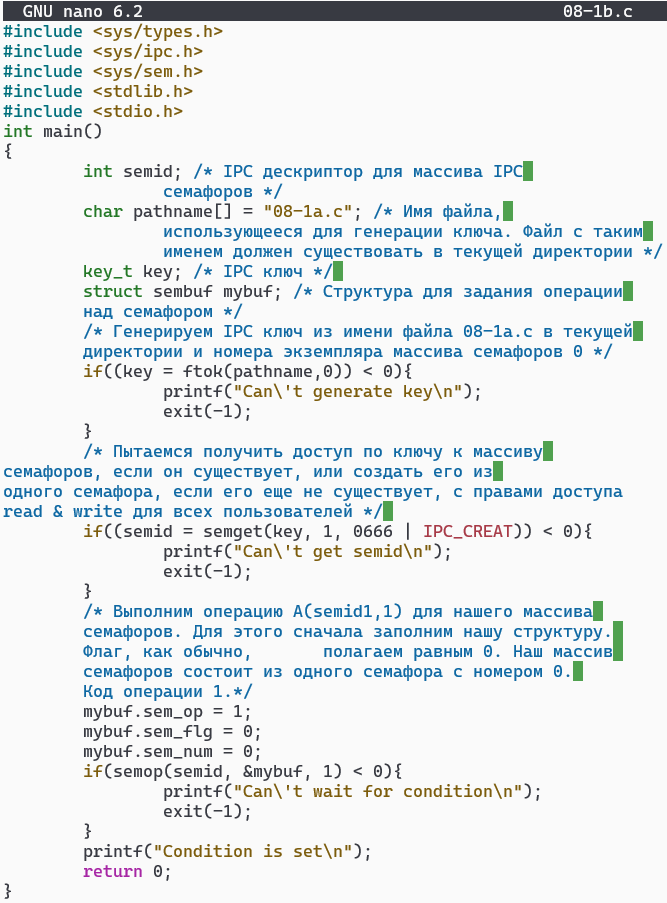


Рисунок 2. Текст программы 08-1b.с



Рисунок 3. Блокировка процесса после запуска программы 08-1а



Рисунок 4. Запуск программы 08-1b



Рисунок 5. Программа 08-1а завершилась

2. Программы 08-1а.с и 08-1b.c были модифицированы таким образом, что первая программа могла работать без блокировки только при условии, что кол-во запусков второй программы было как минимум 5. Это достигается тем, что в первой программе значение sem\_op было -5 (D(S, 5)), таким образом кол-во запусков первой программы без блокировки была равна (кол-ву запусков второй программы) / 5. (Рисунок 6, 7, 8)

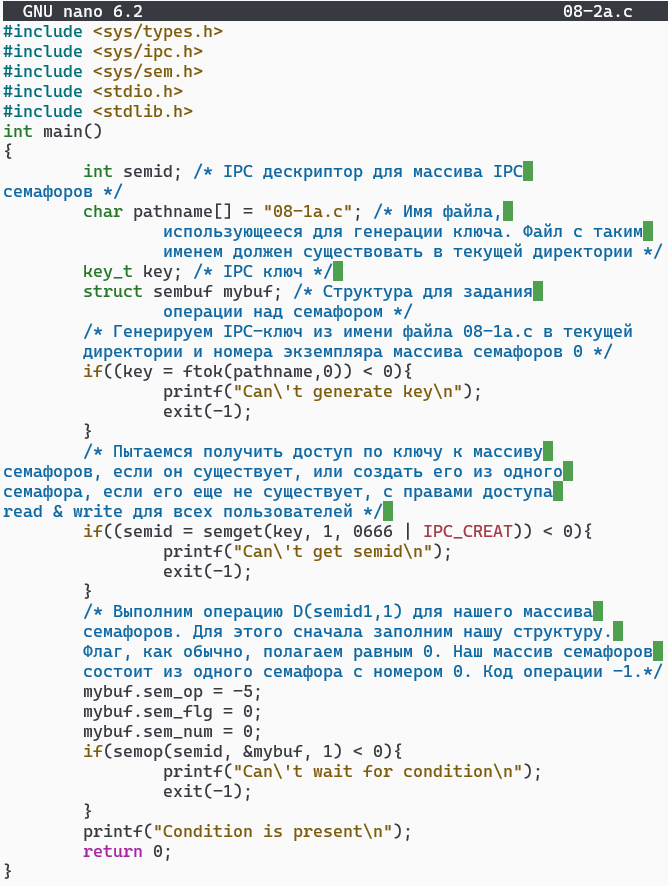


Рисунок 6. Текст доработанной программы 08-1а.с

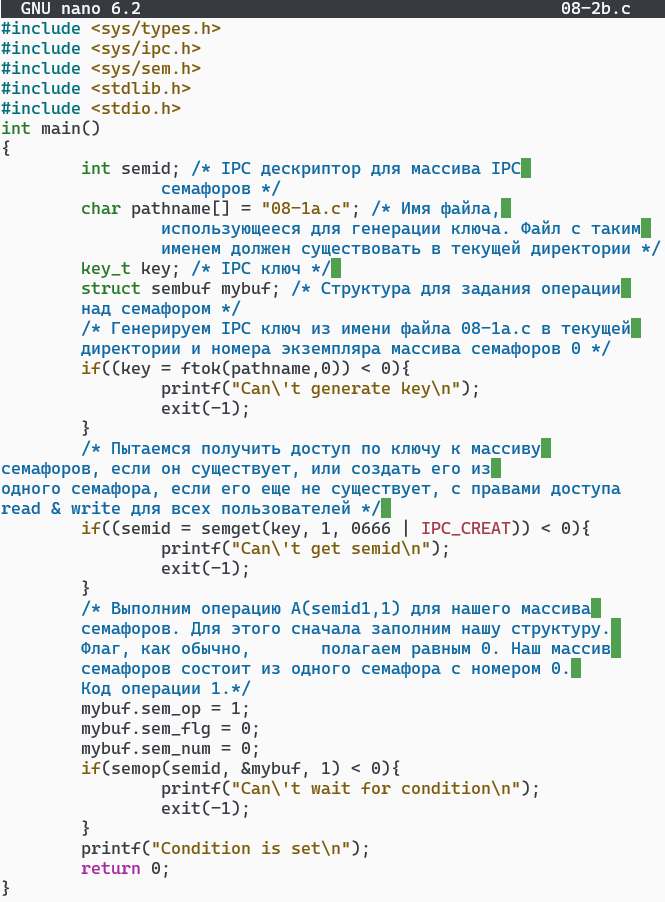


Рисунок 7. Текст доработанной программы 08-1b.с

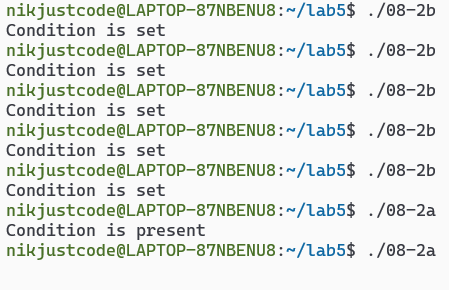


Рисунок 8. Проверка на выполнение условию задания

3. При помощи команды ipcs можно узнать текущие активные семафоры. И, зная значение semid, при помощи команды ipcrm sem можно удалить семафор с указанным semid. (Рисунок 9)

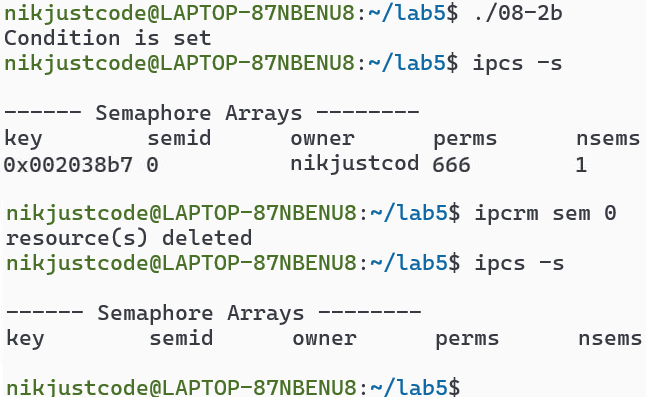


Рисунок 9. Результат использования ipcs и ipcrm

4. Чтобы правильно модифицировать программы 06-3а.с и 06-3b.c, нужно использовать семафор, который будет отвечать за то, можно ли в текущий момент времени какому-либо процессу войти в общую критической секцию. Для правильной работы с семафором можно сразу сделать две переменные struct sembuf, которые будут отвечать за блокировку семафора, то есть доведя значение семафора до 0 (D(S, 1)), и разблокировку, то есть установить значение семафора равным 1 (A(S,1)). Процессы при попытке входа в критическую секцию будут проверять значения семафора и если он равен 0, то процесс заблокируется, пока не случится обратное. (Рисунок 10, 11, 12, 13, 14, 15)

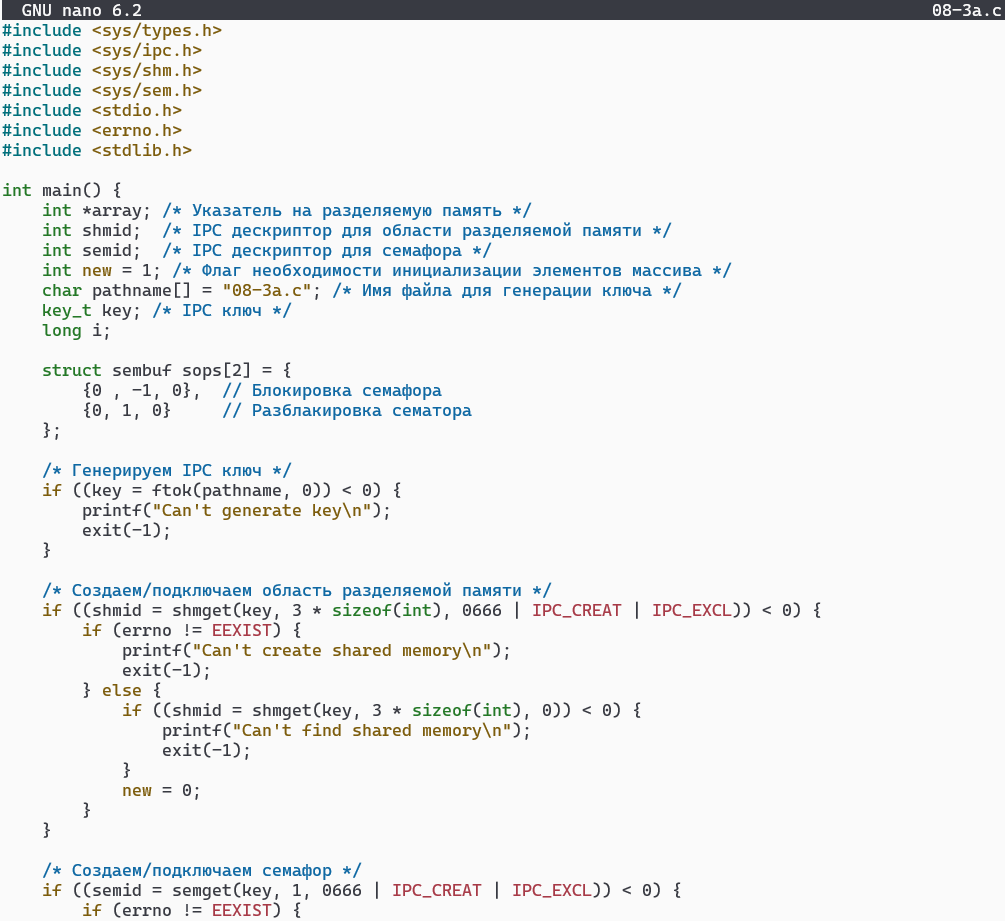


Рисунок 10. Текст модифицированной программы 06-3а.с (Начало)



Рисунок 11. Текст модифицированной программы 06-3а.с (Конец)

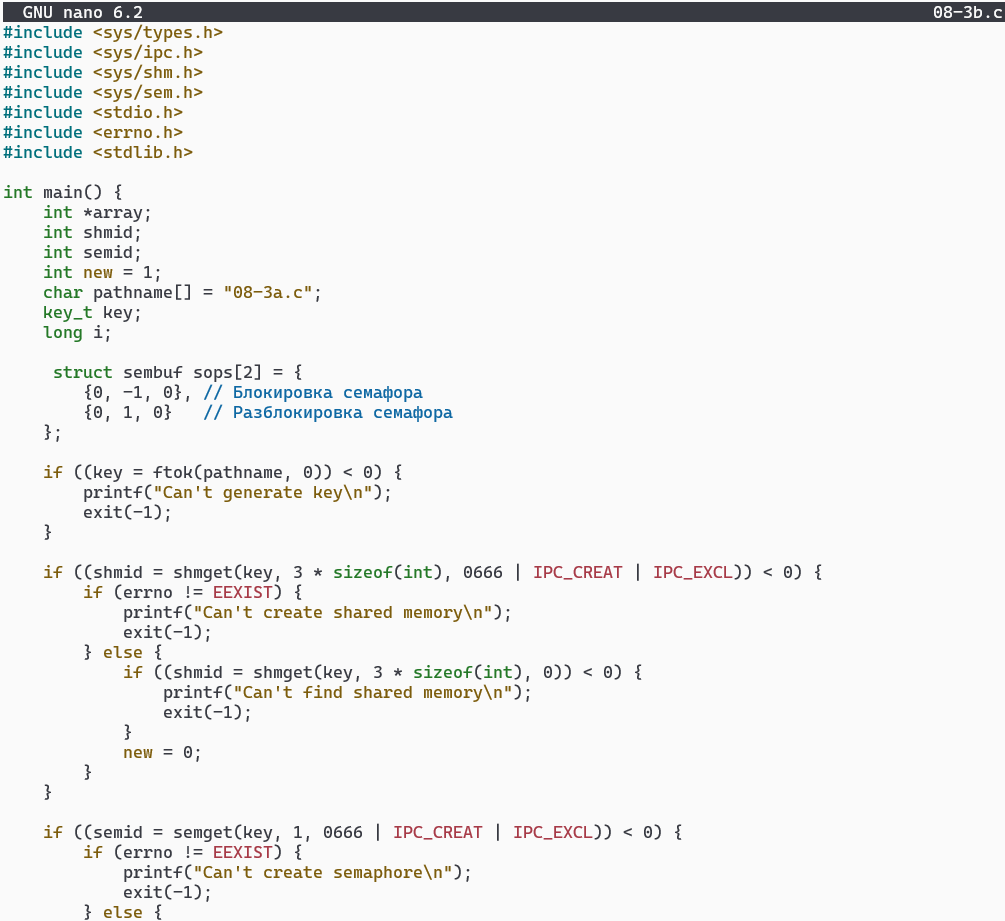


Рисунок 12. Текст модифицированной программы 06-3b.с (Начало)



Рисунок 13. Текст модифицированной программы 06-3b.с (Конец)

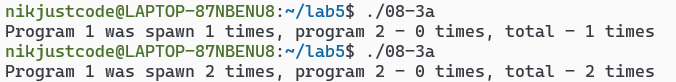


Рисунок 14. Запуск программы 08-3а

C:\Users\1\AppData\Local\Microsoft\Windows\INetCache\Content.Word\11.png

Рисунок 15. Параллельный запуск программы 08-3b

5. Чтобы правильно организовать работу двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe, нужно использовать 2 семафора, каждый из который будет отвечать либо за ребёнка, либо за родителя, то есть можно ли текущему процессу произвести чтение из pipe. Чтобы синхронизировать работу с записью, процессы строго чередуют процесс записи и чтения, поэтому всегда при записи чего-то в pipe одним процессом, второй будет заблокирован, до получения разрешение на продолжение. (Рисунок 16, 17, 18, 19)

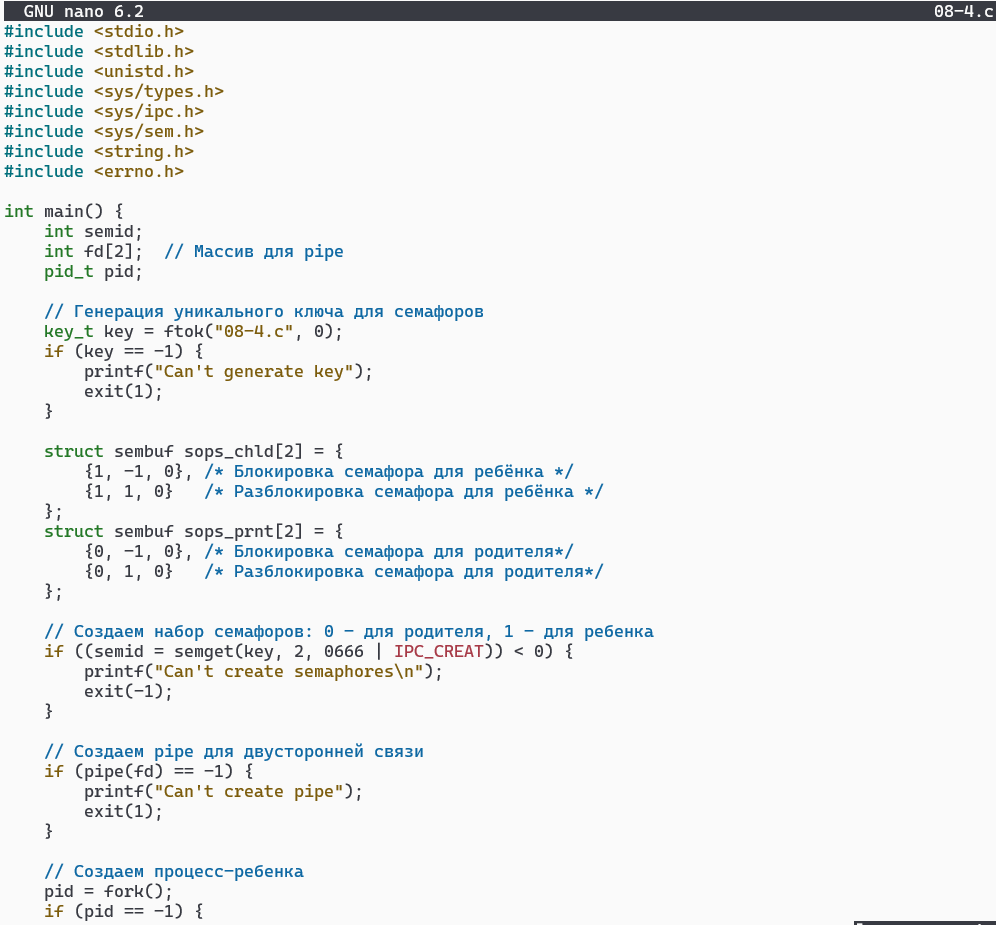


Рисунок 16. Текст программы, использующей семафоры для синхронизации двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe (Начало)

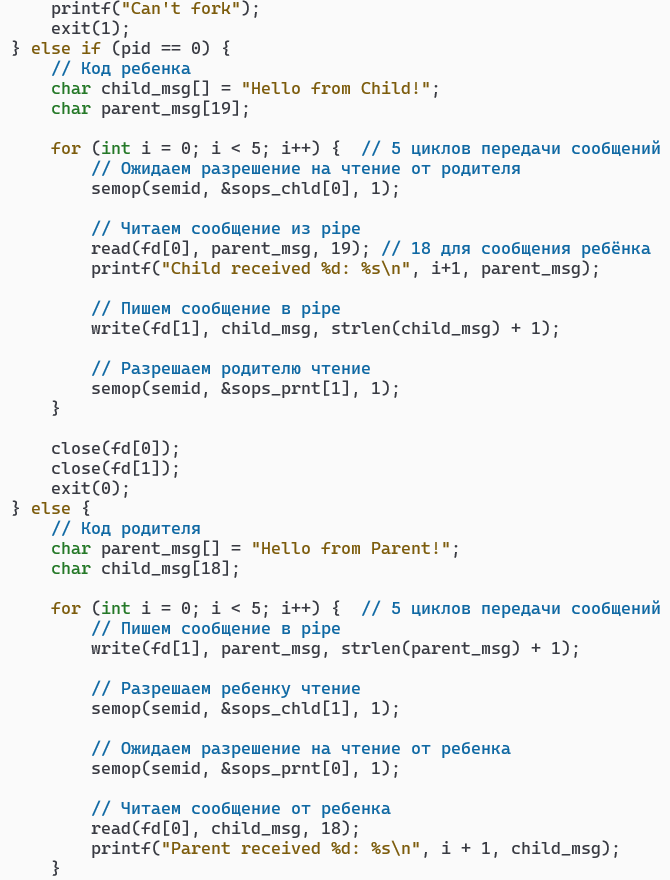


Рисунок 17. Текст программы, использующей семафоры для синхронизации двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe (Продолжение)

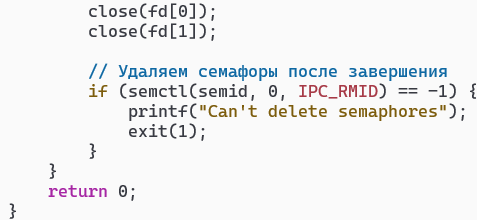


Рисунок 18. Текст программы, использующей семафоры для синхронизации двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe (Конец)

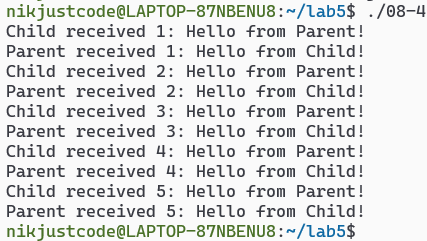


Рисунок 19. Вывод программы, использующей семафоры для синхронизации двусторонней поочередной связи процесса-родителя и процесса-ребенка через pipe

6. В ходе выполнения лабораторной работы были рассмотрены основные механизмы синхронизации процессов при помощи в UNIX. Семафоры в UNIX являются ключевым механизмом для синхронизации процессов, позволяющим надежно организовать управление доступом к общим ресурсам и предотвратить гонки данных. Это особенно важно для многозадачных систем, где параллельная обработка критична для производительности и безопасности. В отличие от классических подходов, UNIX-семафоры поддерживают более гибкие операции: увеличение (A), уменьшение (D) и ожидание обнуления (Z) значения семафора, что позволяет точно контролировать количество доступных ресурсов.

Создание и управление массивом семафоров в UNIX осуществляется с помощью системного вызова semget(), который позволяет использовать уникальный ключ для создания или доступа к массиву семафоров. Это решение упрощает работу с группами семафоров, что особенно полезно для сложных задач синхронизации. Выполнение операций над массивом семафоров происходит с помощью вызова semop(), поддерживающего атомарное выполнение нескольких операций одновременно. Такой подход позволяет процессам гибко регулировать доступ к ресурсам и ожидать их освобождения при необходимости.

Для корректного освобождения ресурсов семафоры можно удалить с помощью команды ipcrm или системного вызова semctl() с флагом IPC\_RMID, что исключает утечки памяти и проблемы при завершении программы.

Таким образом, UNIX-семафоры обеспечивают мощный и гибкий инструмент для управления процессами и ресурсами, позволяя оптимально распределять доступ к системным ресурсам и повышать надежность и эффективность работы многозадачных систем.В результате выполнения лабораторной работы было выявлено, что разделяемая память и нити исполнения значительно повышают производительность многозадачных систем, но требуют правильного управления и синхронизации для предотвращения ошибок при параллельной работе процессов и потоков.