Лабораторная работа №4

студента группы ИТ-222

Мокрищева Николая Павловича

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

**Средства System V IPC. Организация работы с разделяемой памятью в UNIX. Понятие нитей исполнения (thread): версия для печати и PDA**

Цель работы: ознакомление с организацией работы с разделяемой памятью в UNIX, с понятием нитей исполнения (thread).

**Содержание работы**

Вариант №7

1. Тексты программ 06-1a.c и 06-1b.c для иллюстрации работы с разделяемой памятью. Копия экрана, подтверждающего правильность выполнения программ.

2. Результаты выполнения в терминале команд ipcs и ipcrm.

3. Тексты программ, осуществляющие взаимодействие через разделяемую память. Первая программа должна создавать сегмент разделяемой памяти и копировать туда собственный исходный текст, вторая программа должна брать оттуда этот текст, печатать его на экране и удалять сегмент разделяемой памяти из системы. Копия экрана, подтверждающего правильность выполнения программы.

4. Текст программы 06-2.c, в которой работают две нити исполнения. Копия экрана, подтверждающего правильность выполнения программы.

5. Текст программы, в которой работают три нити исполнения. Копия экрана, подтверждающего правильность выполнения программы.

6. Тексты программ 06-3a.с и 06-3b.c для иллюстрации некорректной работы с разделяемой памятью. Копия экрана, подтверждающего правильность выполнения программы.

7. Тексты модифицированных программ для корректной работы с разделяемой памятью при помощи алгоритма Петерсона. Копия экрана, подтверждающего правильность выполнения программы.

8. Вывод.

**Ход работы**

Вариант №7

1. Чтобы хранить информацию в разделяемой памяти, создаём ключ, используя название файла программы, и создаём выделенную память в адресном пространстве, после чего будет хранить кол-во запусков программы 06-1а и 06-1b и выводим эту информацию при запуске одной из программ (Рисунок 1, 2, 3, 4, 5)

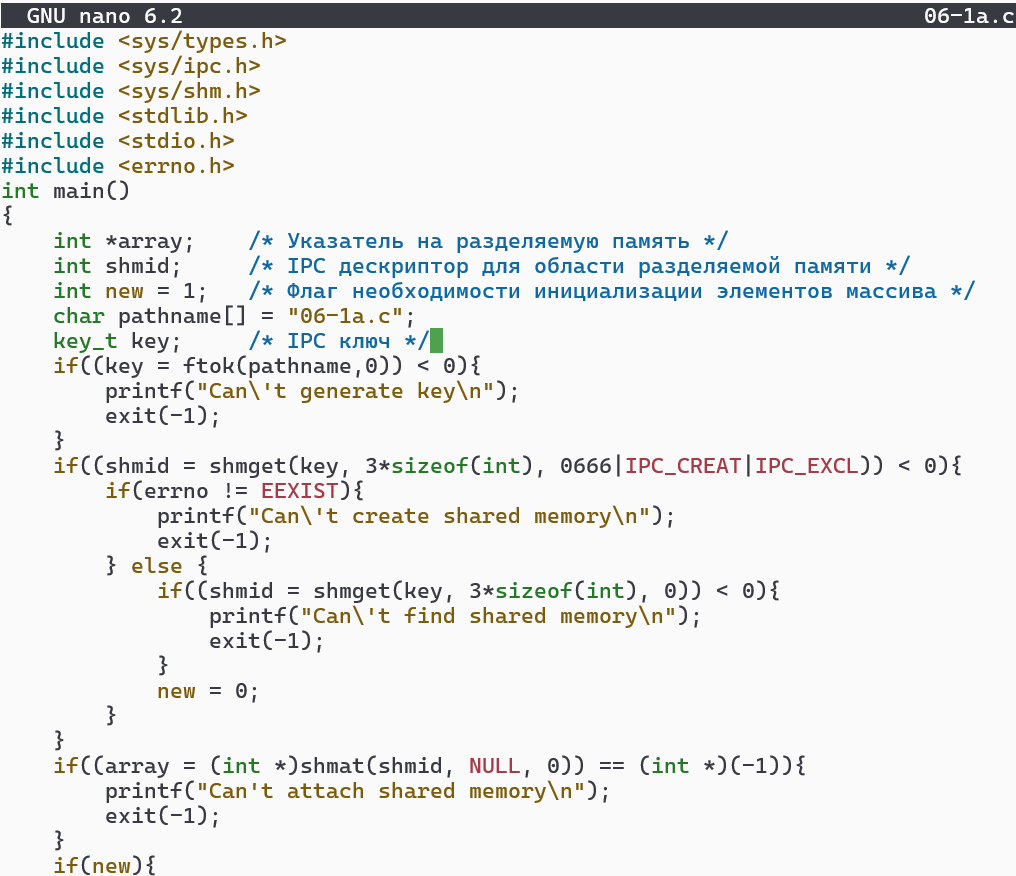


Рисунок 1. Текст программы 06-1а.с (Начало кода)

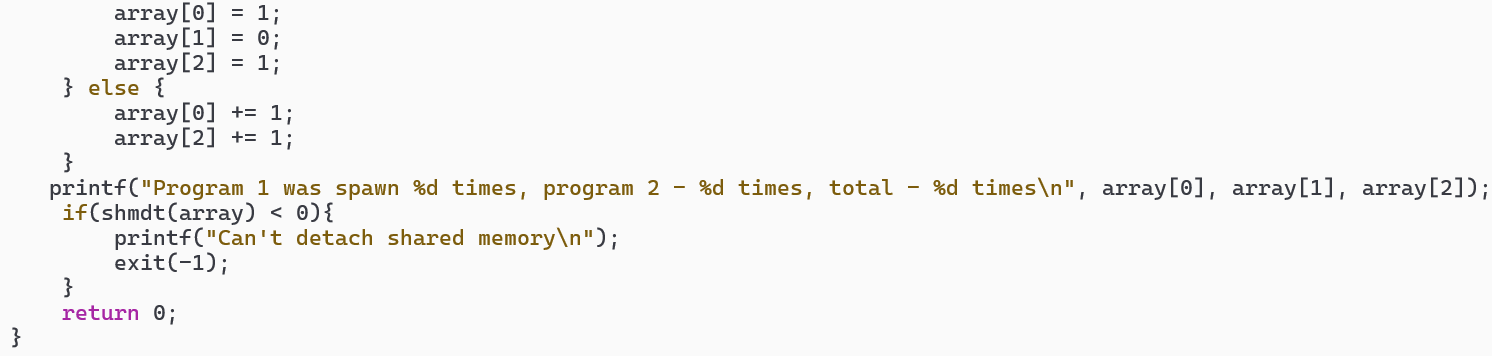


Рисунок 2. Текст программы 06-1а.с (Конец кода)

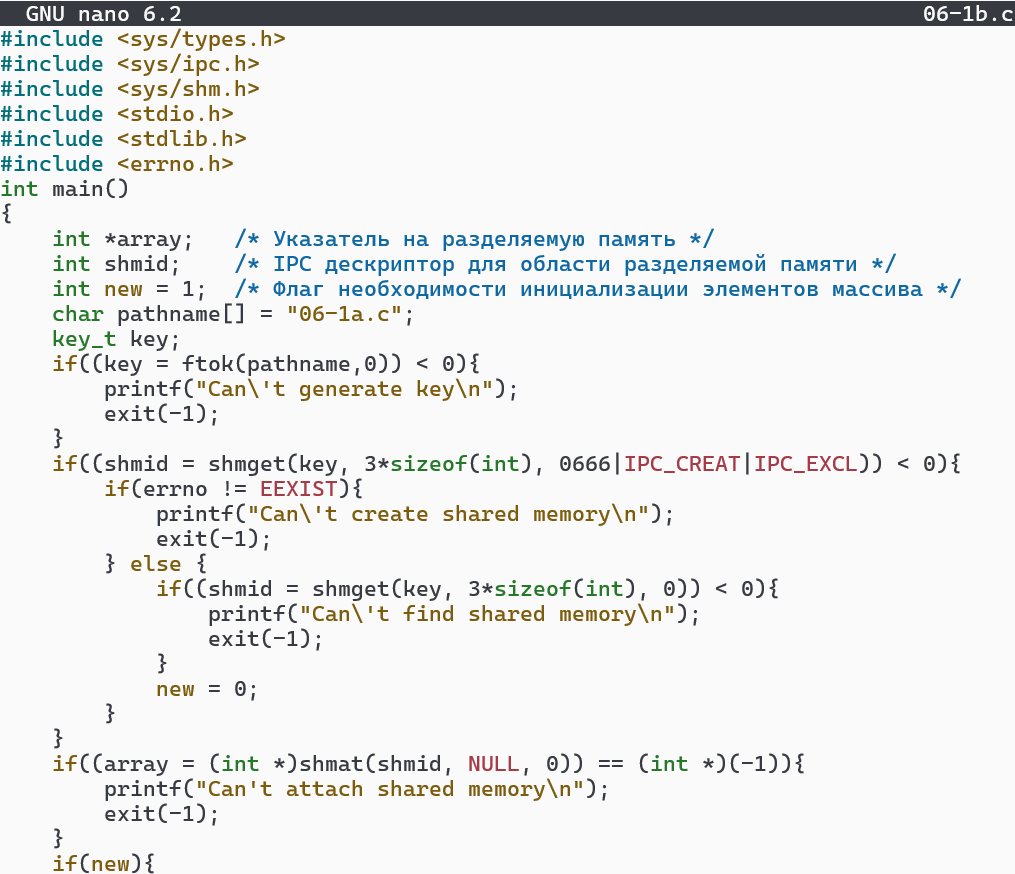


Рисунок 3. Текст программы 06-1b.с (Начало кода)

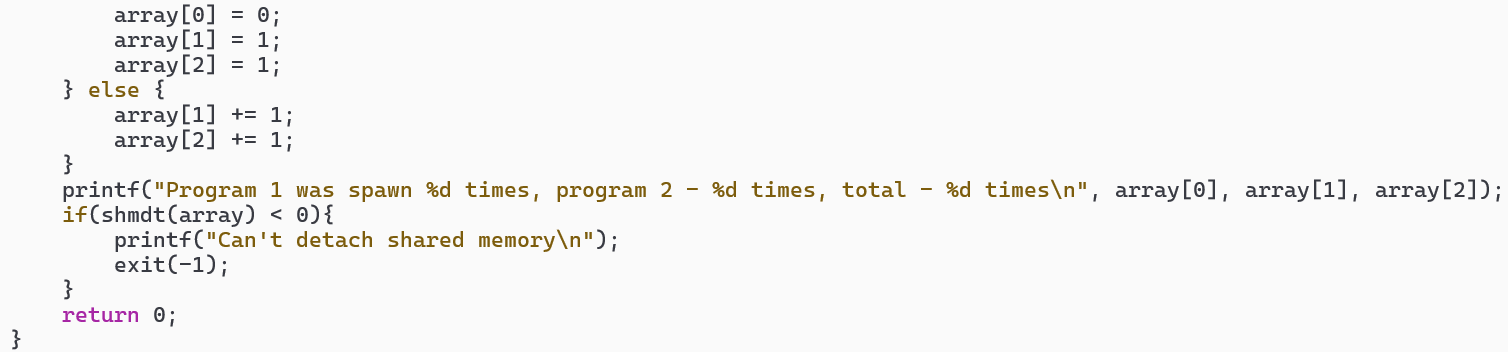


Рисунок 4. Текст программы 06-1b.с (Конец кода)

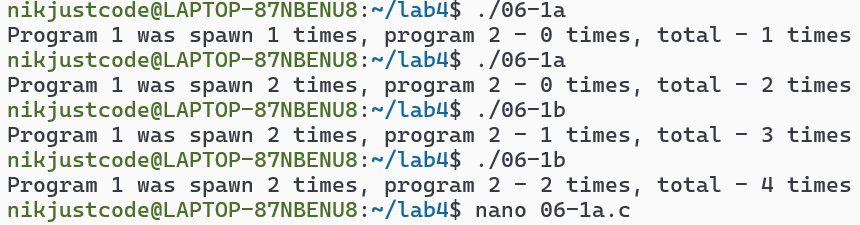


Рисунок 5. Результат после запуска программ

2. При помощи команды ipcs и ключа –m можно вывести информацию о текущих разделяемых памятях, используя дополнительные ключи можно делать данный вывод более направленный, чтобы узнать подробную информацию о времени инициализации памяти, о процессе, который создал её в адресном пространстве и тд. (Рисунок 6)

При помощи команды ipcrm можно удалить разделяемую память, если указать ключ shm. (Рисунок 7)

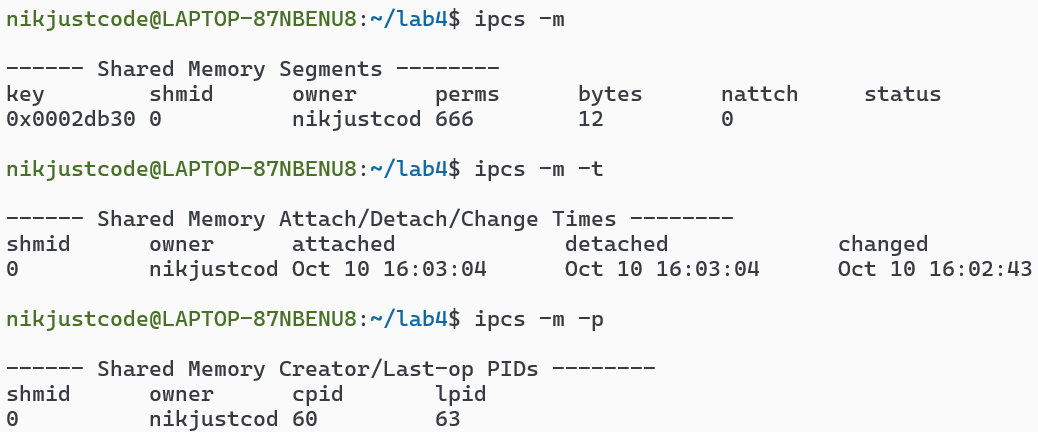


Рисунок 6. Результат выполнения в терминале команды ipcs

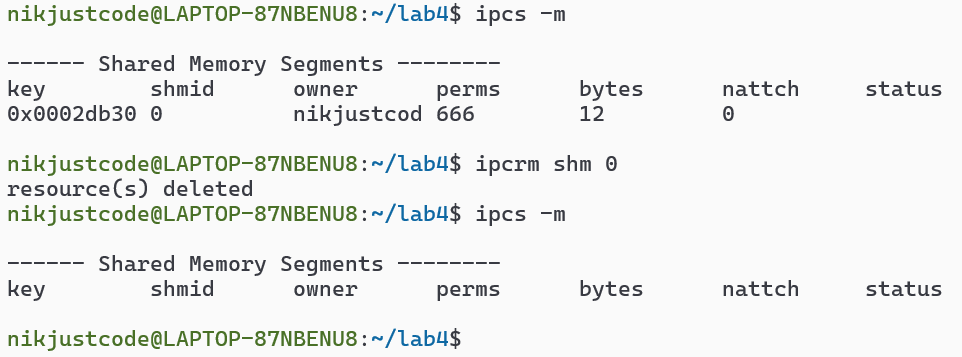


Рисунок 7. Результат выполнения в терминале команды ipcrm

3. Первая программа создаёт сегмент разделяемой памяти и записывает туда сообщение, после чего второй процесс считывает информацию из разделяемой памяти, выводит его в терминал и используя системный вызов shmctl удаляет данную память, чтобы система освободила ресурсы, связанные с данным сегментом разделяемой памяти. (Рисунок 8, 9, 10, 11, 12)

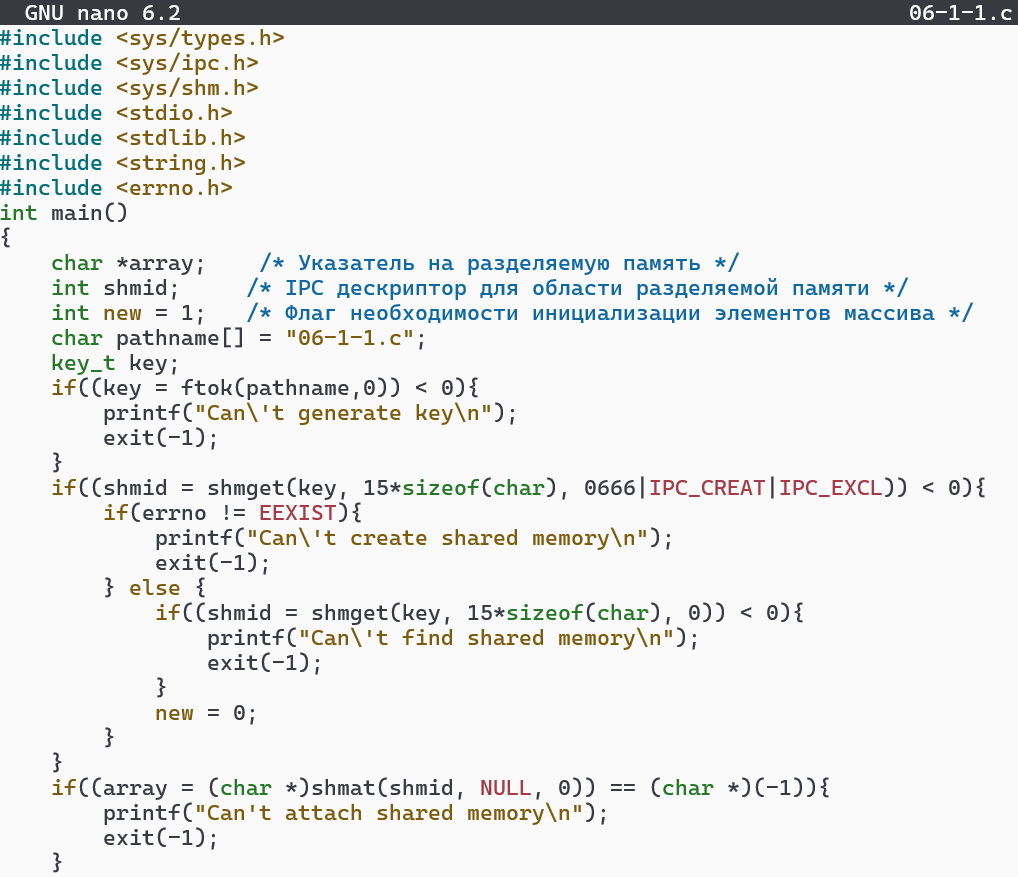


Рисунок 8. Текст программы, передающей сообщение через разделяемую память (Начало)



Рисунок 9. Текст программы, передающей сообщение через разделяемую память (Конец)

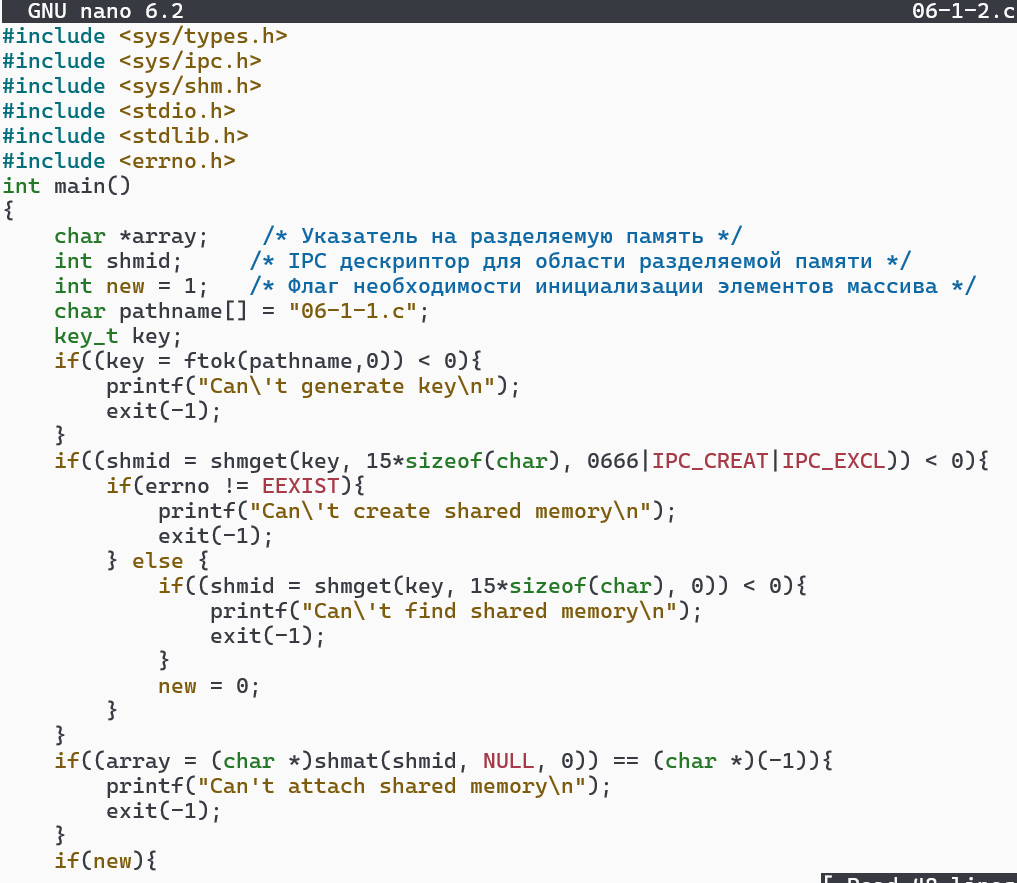


Рисунок 10. Текст программы, получающее сообщение через разделяемую память (Начало)

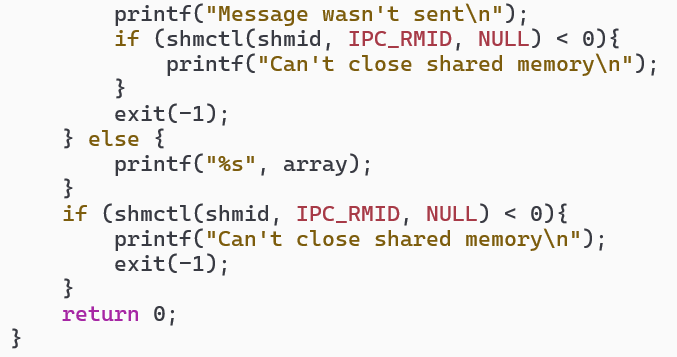


Рисунок 11. Текст программы, получающее сообщение через разделяемую память (Конец)

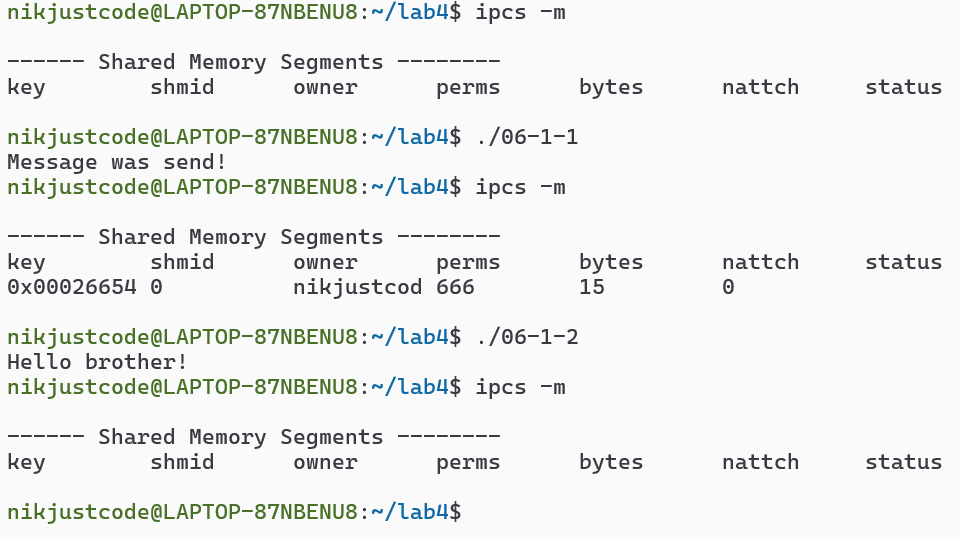


Рисунок 12. Результат запуска программ

4. Используя вторую нить исполнения мы отдельно внутри программы запускаем второй процесс, который выполняет функцию mythread. (Рисунок 13, 14)

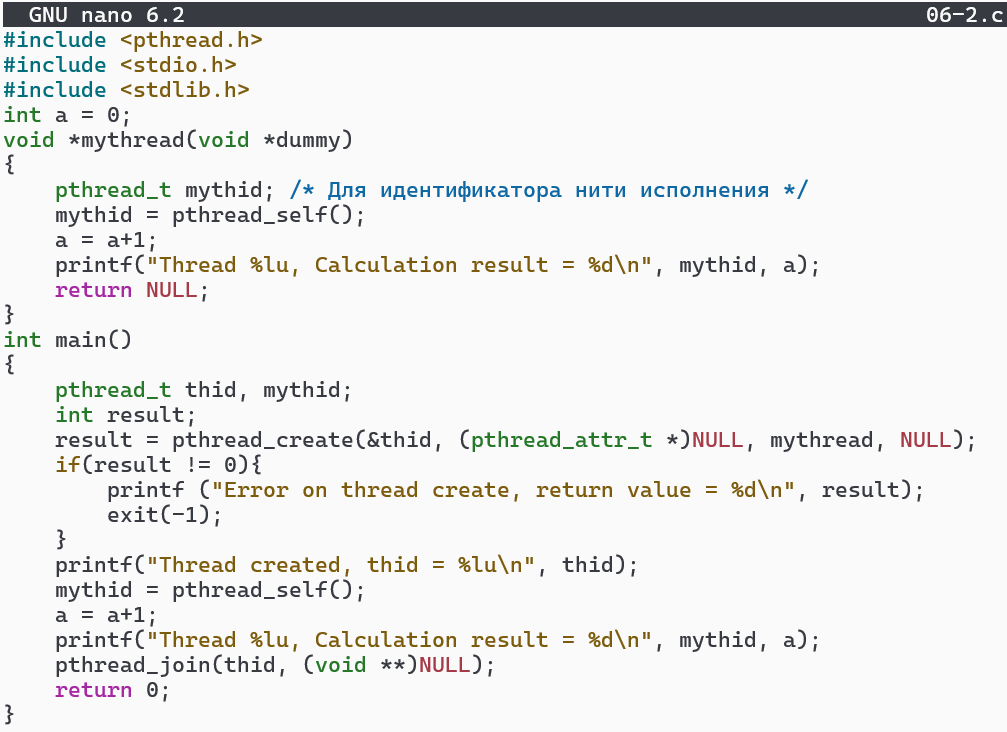


Рисунок 13. Текст программы 05-3.с

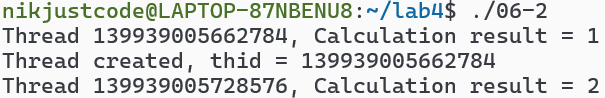


Рисунок 14. Вывод программы 05-3.с

5. В данной программе мы запускаем 3 нити исполнения, каждая из которых выполняет разные действия над одной общей переменной. (Рисунок 15, 16, 17)

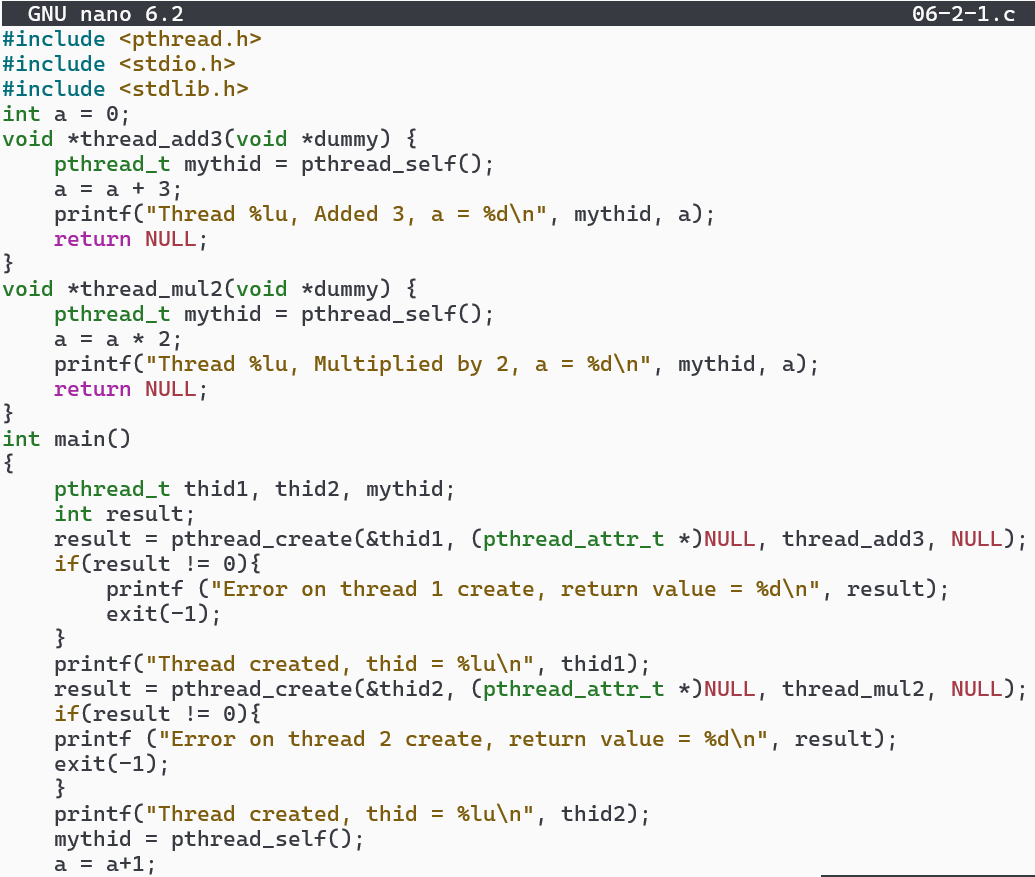


Рисунок 15. Текст программы, запускающий 3 нити исполнения (Начало)

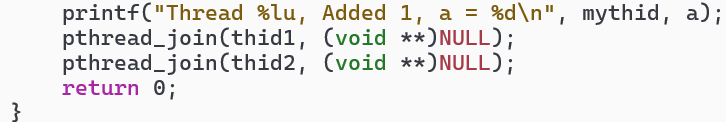


Рисунок 16. Текст программы, запускающий 3 нити исполнения (Конец)

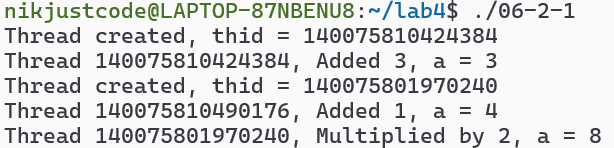


Рисунок 17. Вывод программы, запустивший 3 нити исполнения

6. При запуске двух программ одновременно, они одновременно попадают в критическую секцию, из-за чего нельзя контролировать какой будет вывод у программ, так как они одновременно совершают действия с одинаковыми данными. (Рисунок 18, 19, 20, 21, 22)



Рисунок 18. Текст программы 06-3а.с (Начало)

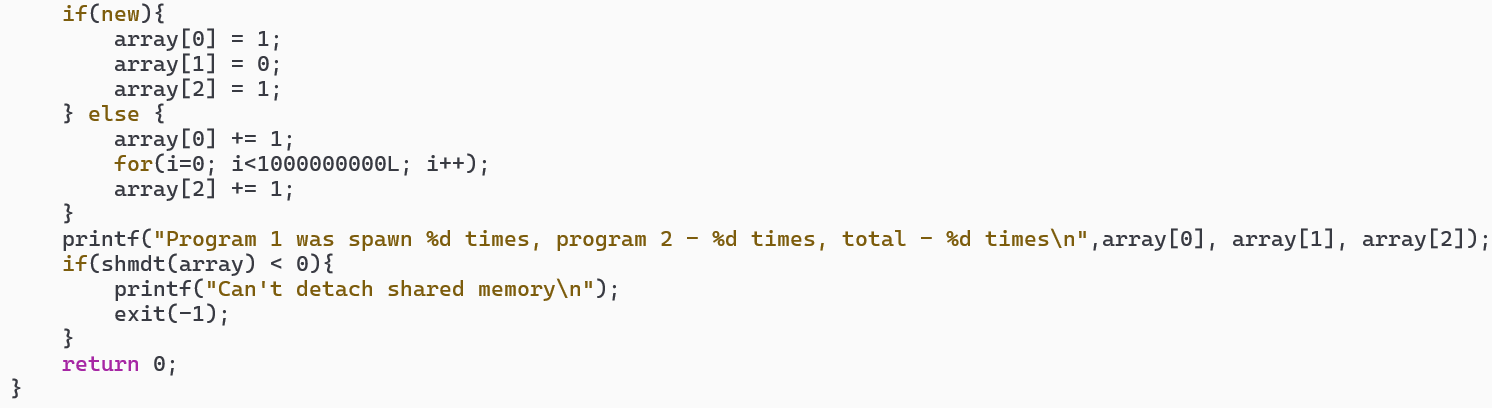


Рисунок 19. Текст программы 06-3а.с (Конец)



Рисунок 20. Текст программы 06-3b.с (Начало)

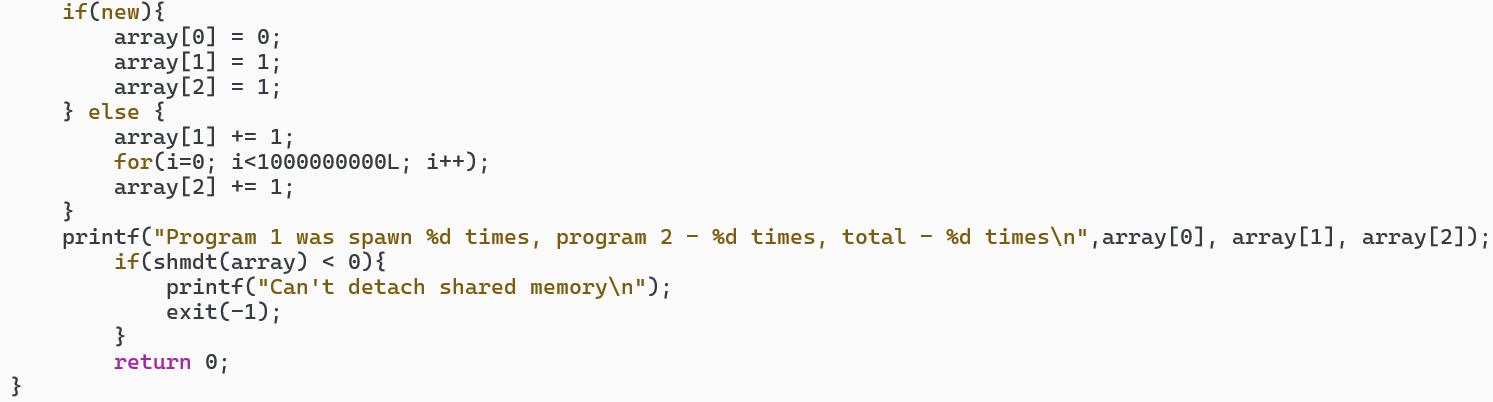


Рисунок 21. Текст программы 06-3b.с (Конец)

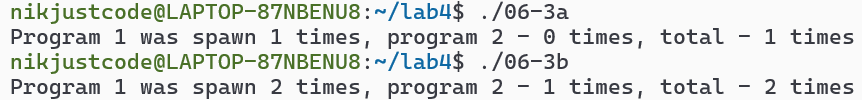


Рисунок 22. Вывод программ при одновременном запуске

7. Чтобы исправить проблему из предыдущего задания, используя алгоритм Петерсона, нужно, чтобы две программы никогда одновременно не вошли в критическую секцию, для этого мы добавляем две переменные, которые будут говорить о том, могут ли программы входить в критическую секцию и какой процесс следующим сможет это сделать. (Рисунок 23, 24, 25, 26, 27)

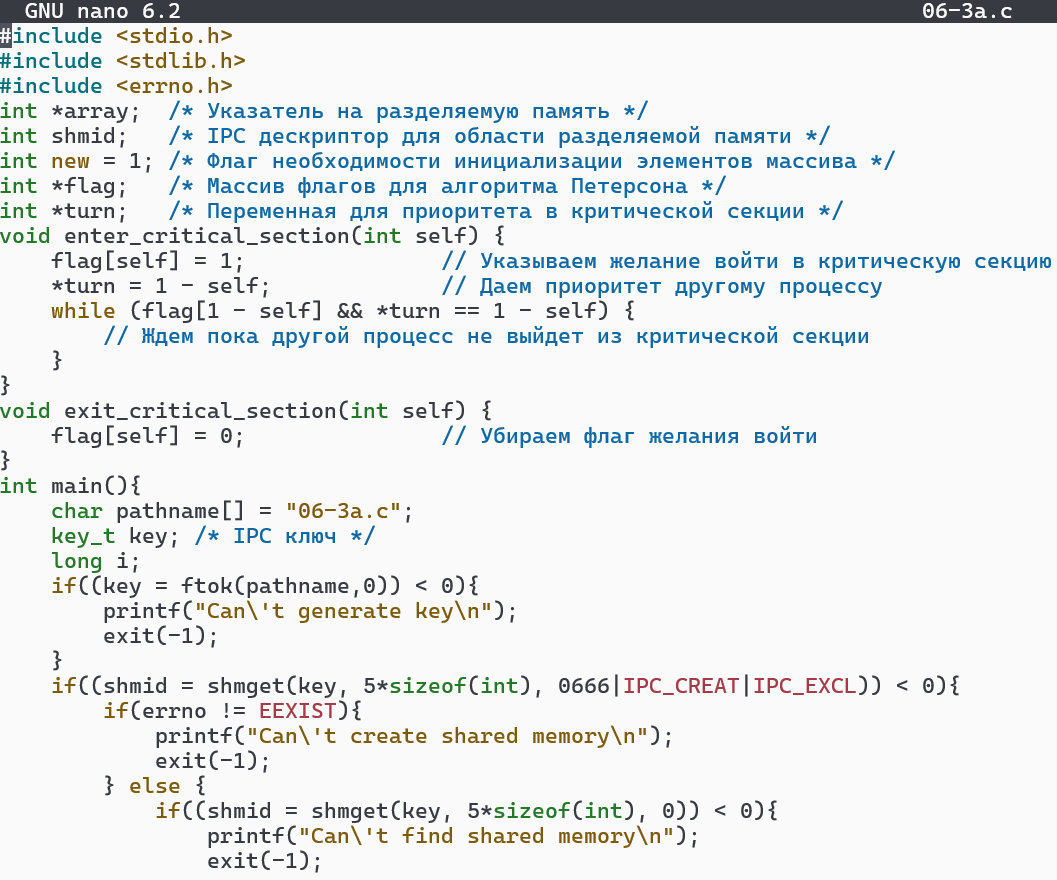


Рисунок 23. Текст исправленной программы 06-3а.с (Начало)

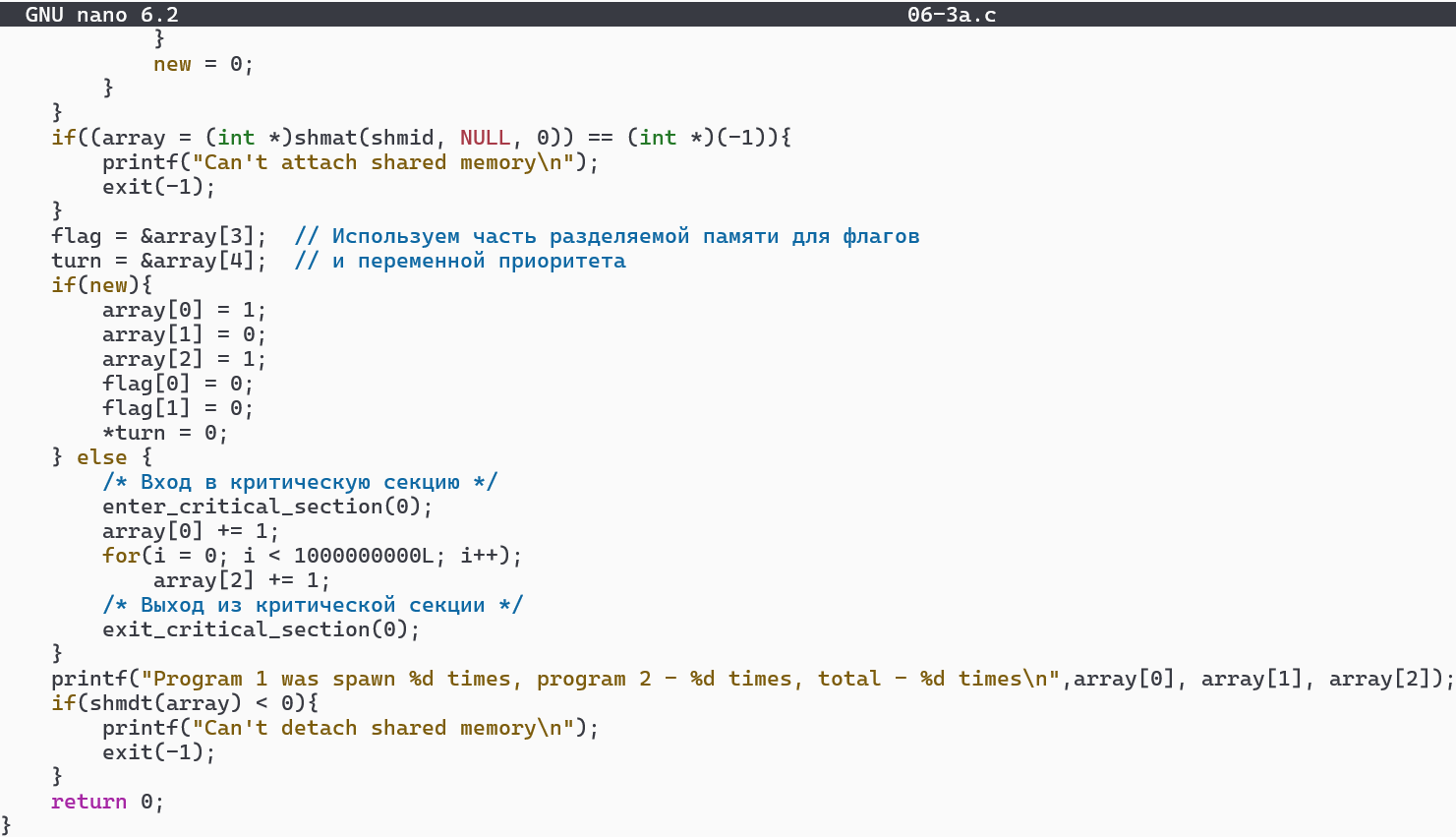


Рисунок 24. Текст исправленной программы 06-3а.с (Конец)

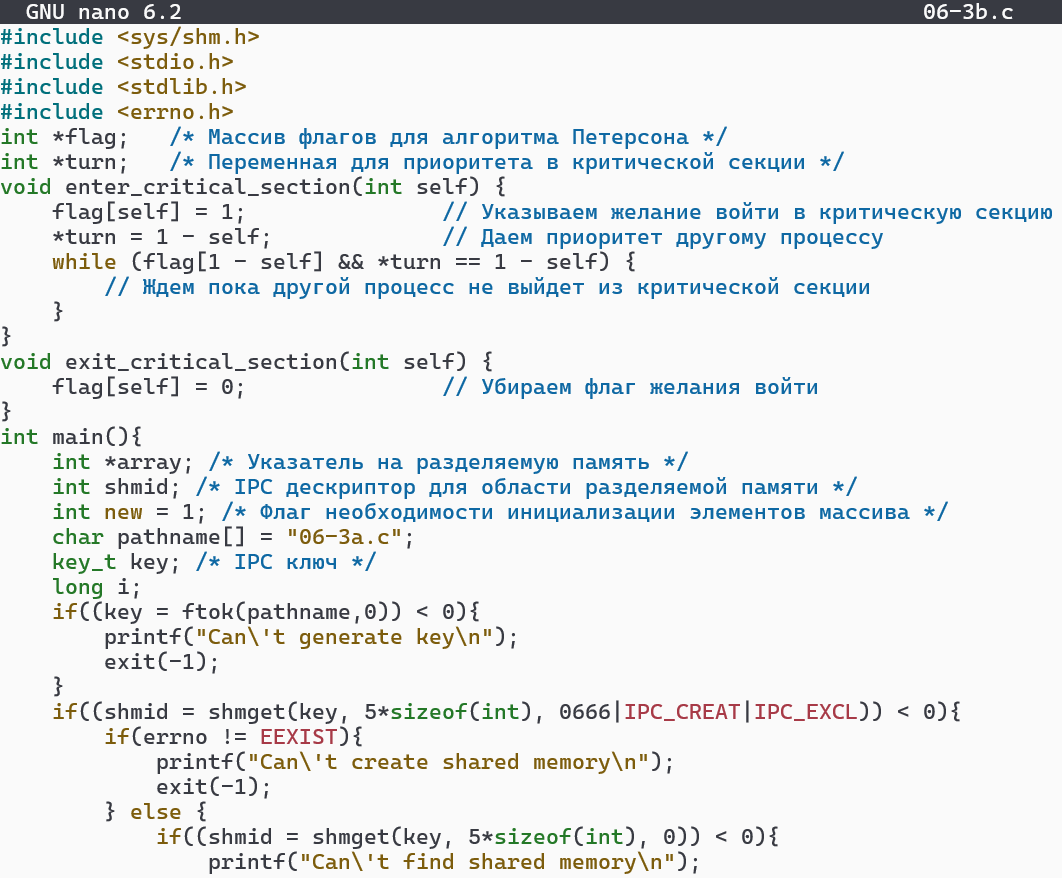


Рисунок 25. Текст исправленной программы 06-3b.с (Начало)

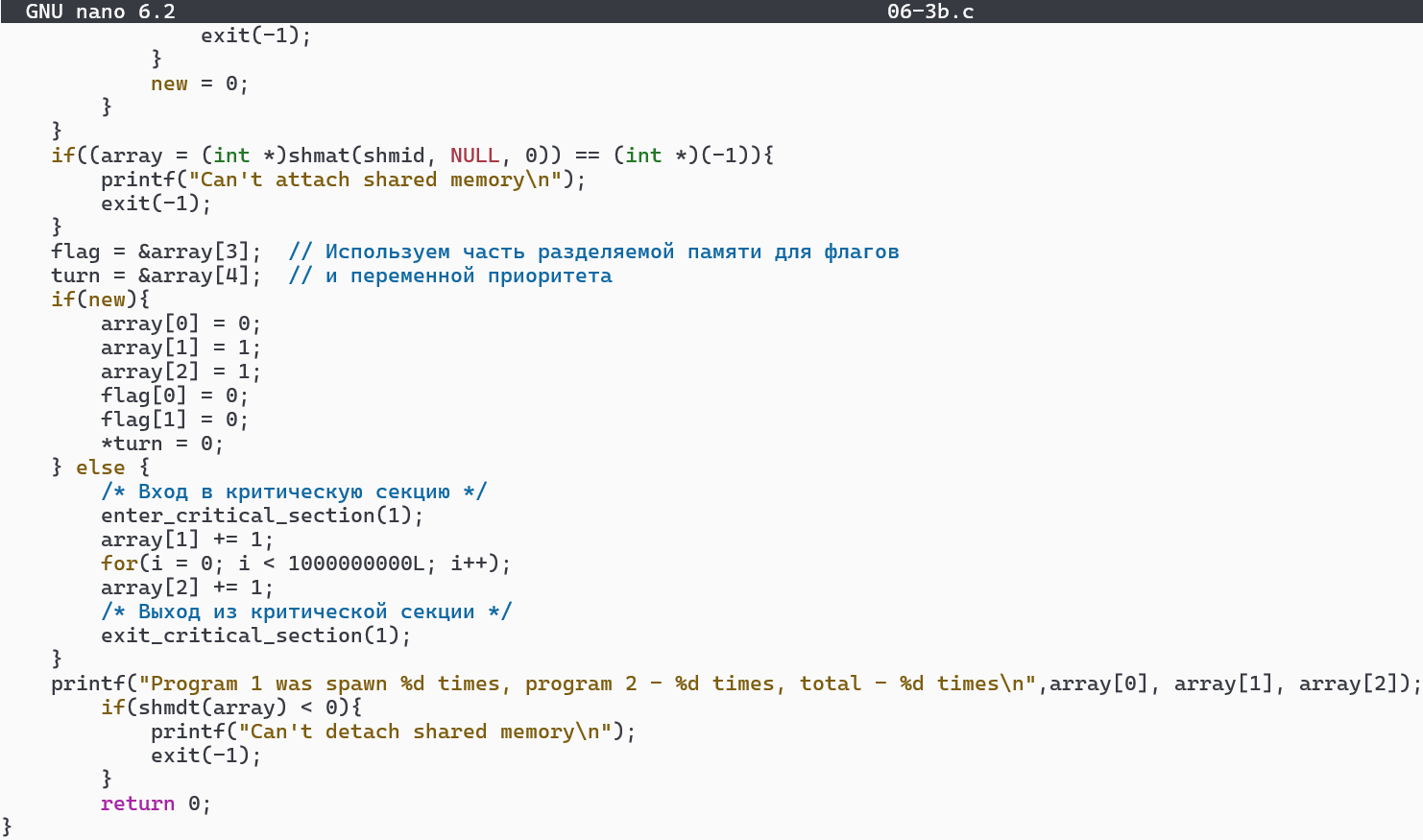


Рисунок 26. Текст исправленной программы 06-3b.с (Конец)

C:\Users\1\AppData\Local\Microsoft\Windows\INetCache\Content.Word\18.2.png

Рисунок 27. Корректный вывод после работы программы

8. В ходе выполнения лабораторной работы были рассмотрены основные механизмы межпроцессного взаимодействия в UNIX на основе средств System V IPC и нити исполнения (thread). Разделяемая память оказалась эффективным способом обмена данными между процессами, позволяя экономить ресурсы, так как данные не копируются между процессами, а разделяются. Основными системными вызовами для работы с разделяемой памятью являются `shmget()`, `shmat()`, `shmdt()` и `shmctl()`. Эти функции позволяют выделять, подключать, отключать и освобождать разделяемую память. Команды `ipc` и `ipcrm` помогают в управлении этими объектами на уровне системы.

Функция `ftok()` была использована для генерации уникального ключа, необходимого для создания и обращения к объектам System V IPC, что обеспечило корректную адресацию ресурсов. Разделяемая память активно взаимодействует с вызовами `fork()`, `exec()` и `exit()`, что важно для процессов, использующих общий сегмент памяти.

Также была изучена концепция нитей исполнения (thread) в UNIX, которая позволяет параллельно выполнять несколько потоков внутри одного процесса. Это снижает накладные расходы по сравнению с процессами, так как потоки используют общее адресное пространство. Основные функции для работы с нитями включают `pthread\_create()` для создания нити, `pthread\_exit()` для ее завершения и `pthread\_join()` для синхронизации с завершением работы другой нити. Определение текущей нити возможно с помощью функции `pthread\_self()`.

Главным преимуществом потокового обмена данными является возможность одновременного выполнения задач, что ускоряет работу приложений, однако возникают сложности с синхронизацией. Для предотвращения гонок данных необходимо использовать механизмы синхронизации, такие как мьютексы и семафоры.

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.