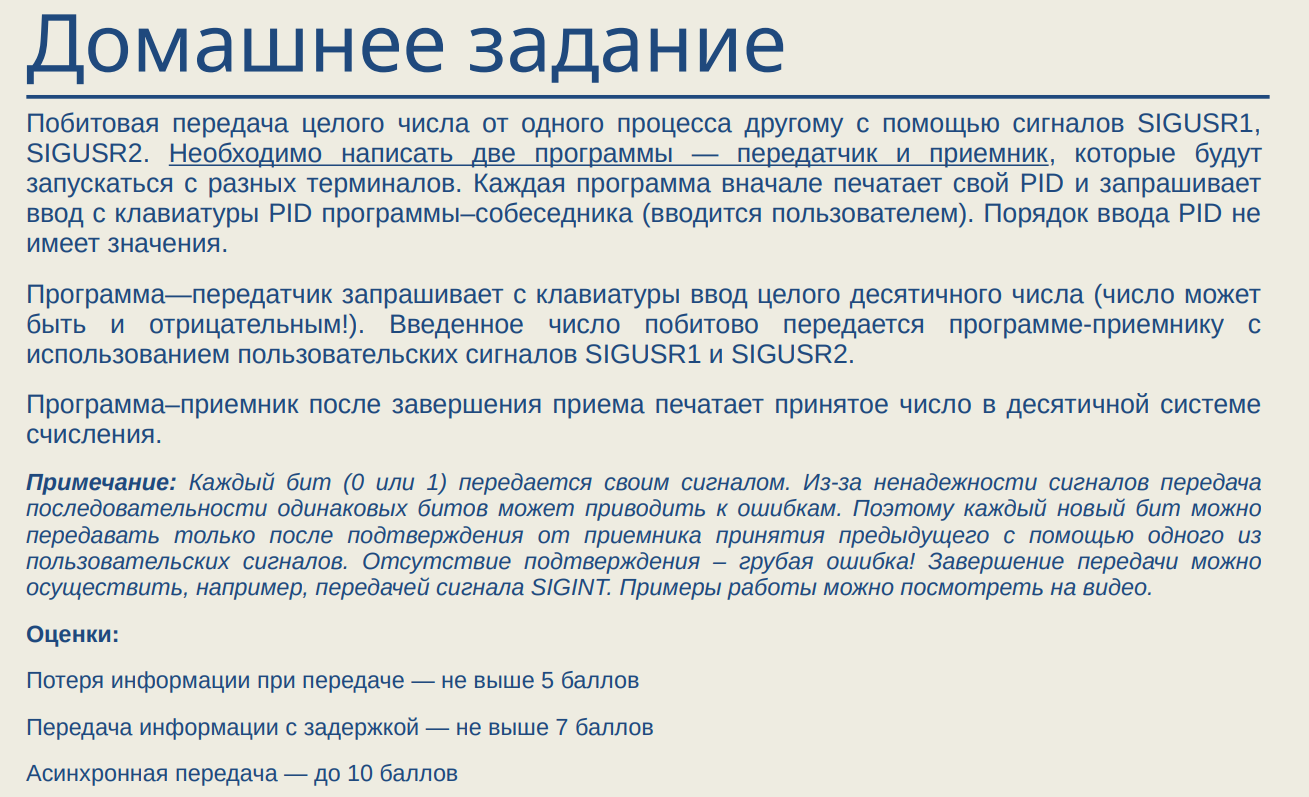
Студент Крамаренко М.К., группа 234, ДЗ №5



### Запуск программы:

1. Скомпилировать программу:

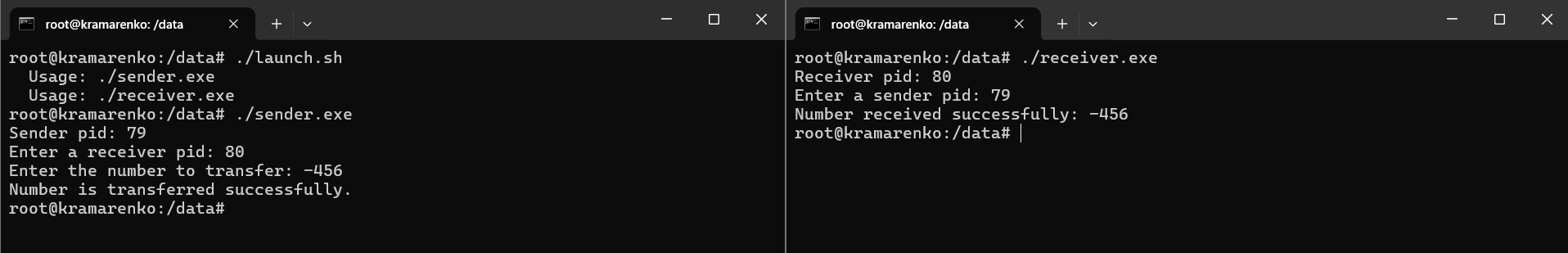
**gcc sender.c -o sender.exe**

**gcc receiver.c -o receiver.exe**

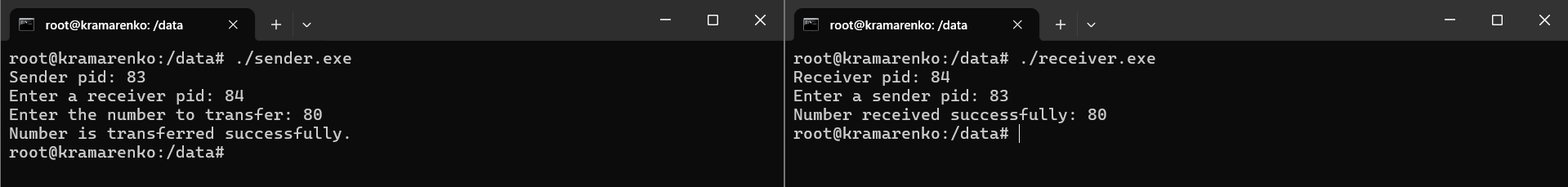
1. Запустить программу:

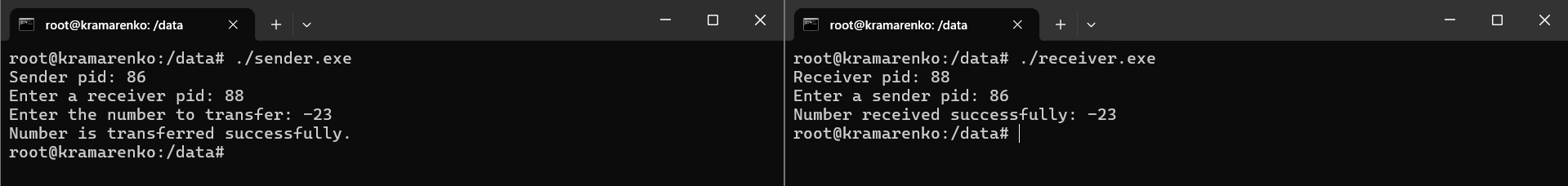
**./sender.exe**

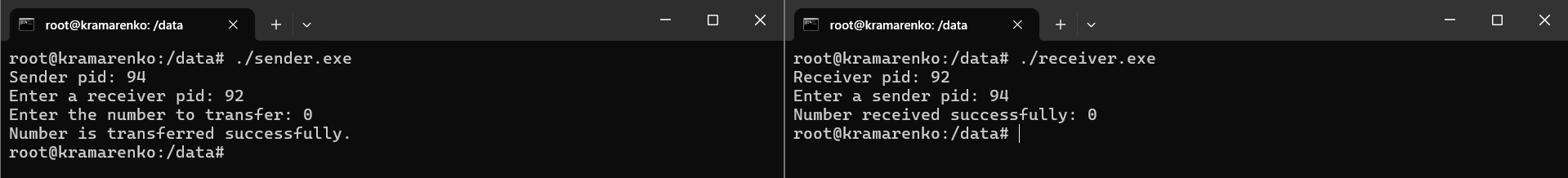
**./receiver.exe**

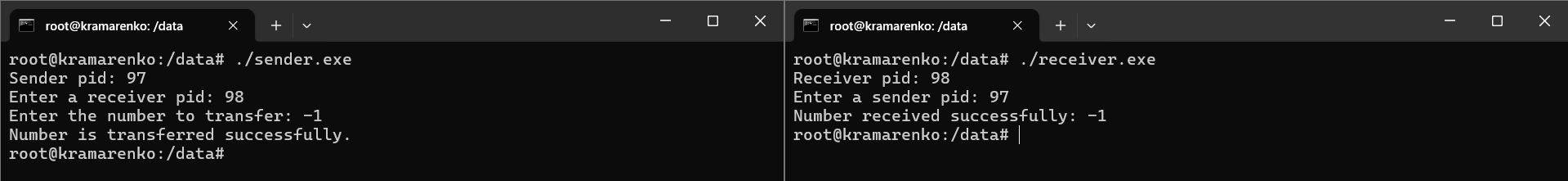


### Пример работы программы:

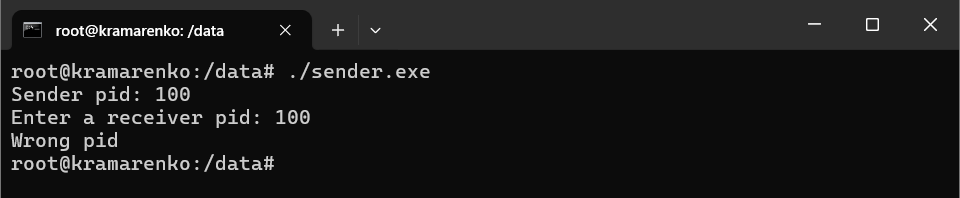


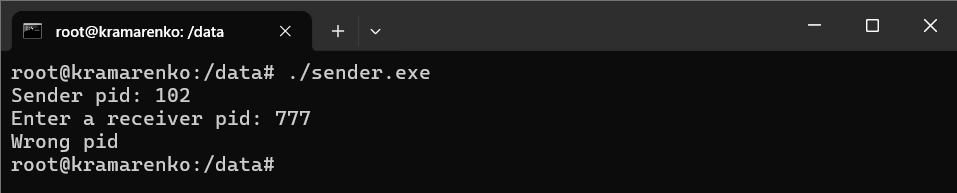


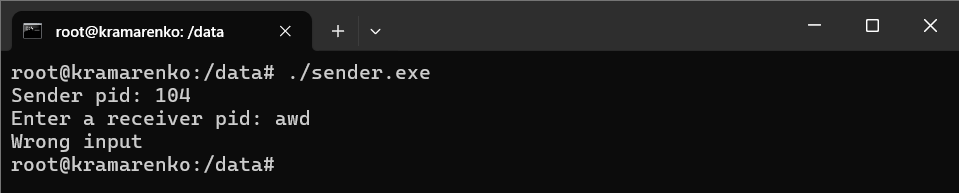




**Некорректный** ввод обрабатывается:



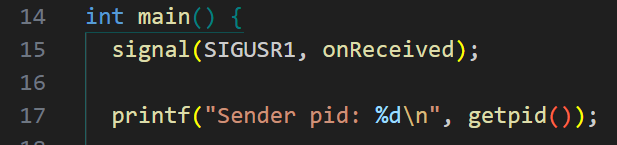




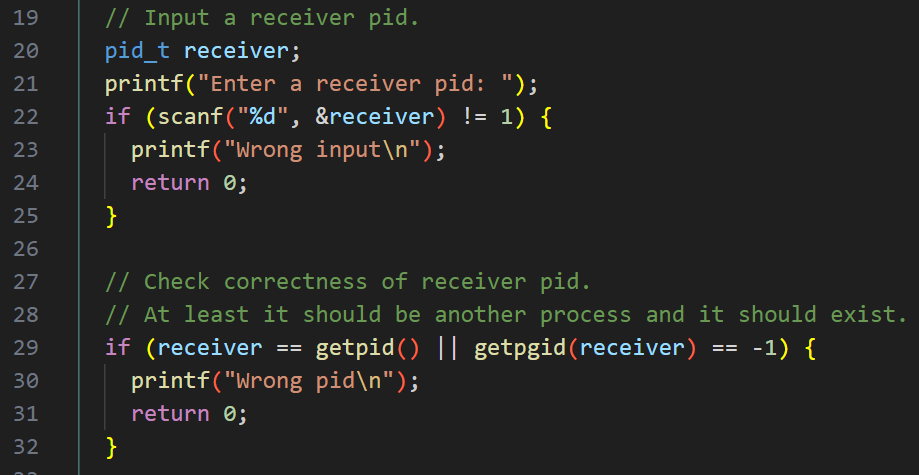
# **Реализация:**

## **sender.c**

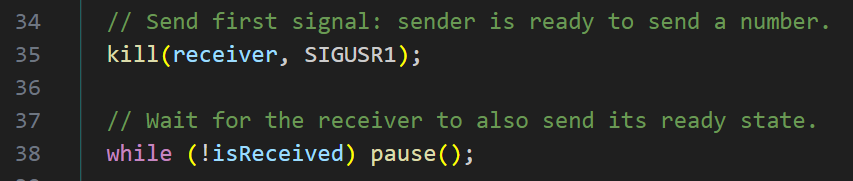
1. Первое – это перехват сигнала **SIGUSR1** и вывод своего номера процесса:



1. Затем я запрашиваю номер процесса получателя и делаю небольшую проверку этого номера:

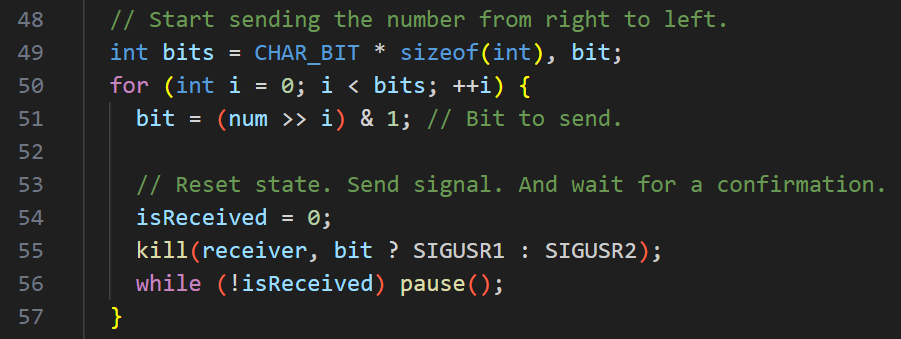


1. Дальше я отправляю первый сигнал, который означает что я готов отправлять число и жду такой же сигнал от получателя (о готовности получать число).



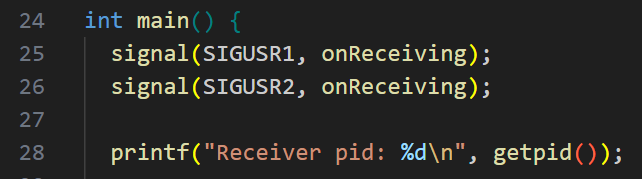
1. Дальше я вычисляю количество бит в типе **int** и побитово отправляю число (справа налево).

Если бит равен **1**, то отправляю сигнал **SIGUSR1**, иначе – **SIGUSR2**.

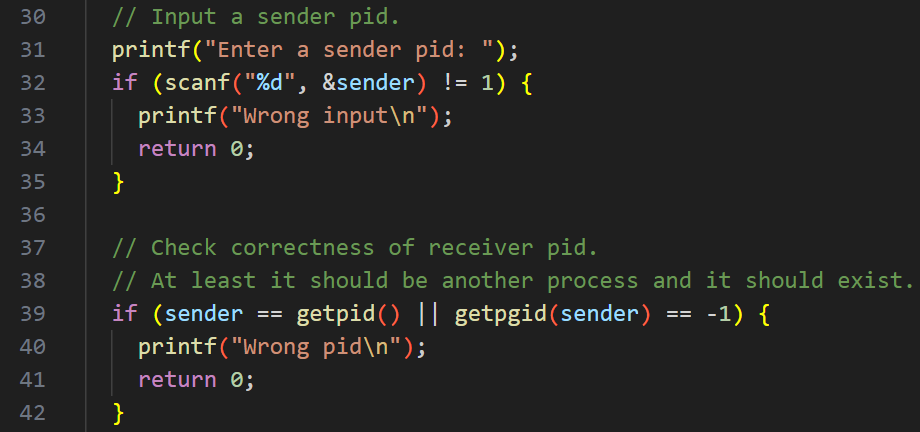


## **receiver.c**

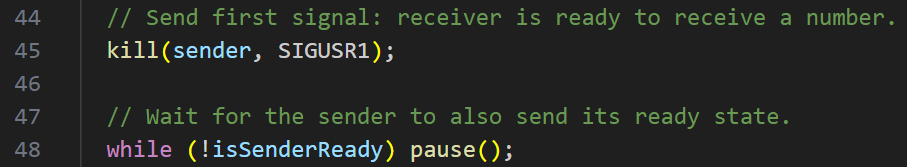
1. Первое – это перехват сигнала **SIGUSR1**, **SIGUSR2** и вывод своего номера процесса:



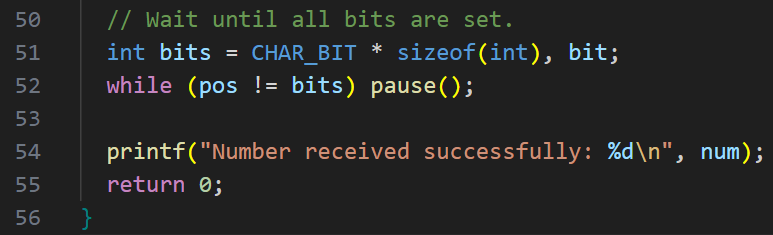
1. Затем я запрашиваю номер процесса отправителя и делаю небольшую проверку этого номера:



1. Дальше я отправляю первый сигнал, который означает что я готов получать число и жду такой же сигнал от отправителя (о готовности отправить число).



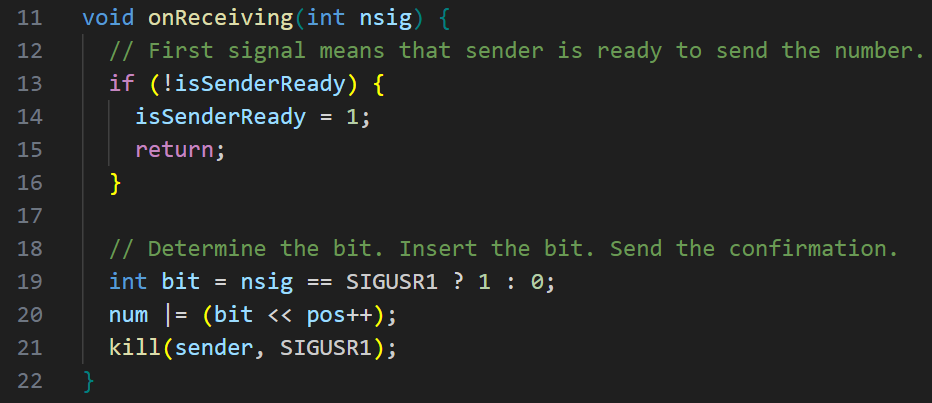
1. Дальше я вычисляю количество бит в типе **int** и жду пока не заполняться все эти биты, после того, как все биты были установлены я вывожу полученное число:



1. При получении сигнала **SIGUSR1** или **SIGUSR2** вызывается функция **onReceiving()**.

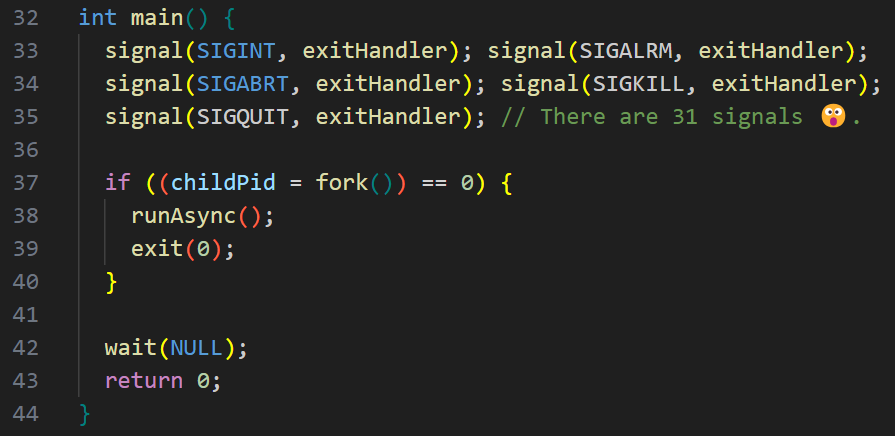
Сначала я жду первый сигнал, который означает, что отправитель готов отправлять число.

Последующие сигналы означают значения битов: **SIGUSR1** – 1, **SIGUSR2** – 0.   
После получения очередного бита, я отправляю сигнал о подтверждении получения бита.



Позже немного засомневался на счёт асинхронности, что именно подразумевалось под этим словом. Изначально мне показалось, что надо было избавиться от цикла while(1);, т.к. даёт сильную нагрузку на CPU. Поэтому я нашёл функцию pause(), которая ожидает какого-либо сигнала.

Но в итоге, чтоб наверняка, создал папку **async**, в ней лежат два файлика **sender.c** и **receiver.c**. Они запускают процессы, которые выполняют логику выше, а сама программа может заниматься чем-то другим (в моём решении, она просто ждёт завершение созданного процесса). А также добавил несколько обработчиков сигналов, для корректного завершения всех процессов:



# **Итого**

Есть две программы которые через системные сигналы побитово передают введённое число.

В целом работа с сигналами для передачи информации очень неудобная, а местами вообще не понятная.

У меня было много классных идей реализации, но заработала только эта. В других реализациях системные сигналы дублировались (из-за чего программа зависала) или сигналы перезаписывались (программа отправляла новый сигнал, а вторая программа получала предыдущий сигнал ещё раз, но общее их количество сигналов оставалось верным (мьютексы не помогли).

Или, например, костыль в этой программе. В **receiver.c** в методе **onReceiving()** есть проверка на первый сигнал. Изначально была идея сделать два обработчика. Когда первый обработчик получает первый сигнал, он сразу устанавливает другой обработчик сигналов, т.е. устанавливает обработчик для последующих вызовов (запись битов). Но в такой реализации всё висло. Пришлось эти два обработчика объединить в один.

Или, например, в **sender.c**, если поменять местами ввод числа и проверку готовности получателя, программа чаще всего работает неправильно, хотя логика не изменилась.

В общем много странных моментов, о которых нигде не пишут.