**Лабораторная работа № 6 – Генерация и обработка сигналов**

**Цель работы**

Освоение простейшего средства управления процессами, позволяющего процессам передавать информацию о каких-либо событиях, отрабатывать реакции на различные события и взаимодействовать друг с другом.

**Пункт 1**

Программа sigint.cpp осуществляет ввод символов со стандартного ввода. Скомпилируйте и запустите программу и отправьте ей сигналы SIGINT (нажатием Ctrl-C) и SIGQUIT (нажатиемCtrl-\). Проанализируйте результаты.

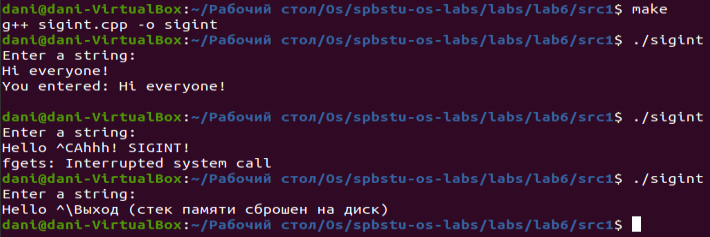


Рис. 6-1 Работа sigint.cpp.

Данная программа принимает на вход строку и выводит её обратно в консоль. Если отправить сигнал SIGINT, то процесс его перехватит и выведет строку “Ahhh! SIGINT!”. Это сделано при помощи системного вызова **sigaction,** который используется для изменения действия процесса при получении соответствующего сигнала. Для сигнала SIGQUIT обработчик не установлен, поэтому он работает в штатном режиме.

**Пункт 2**

Запустите программу signal\_catch.cpp , выполняющую вывод на консоль. Отправьте процессу сигналы SIGINT и SIGQUIT, а также SIGSTOP (нажатием Ctrl-Z) и SIGCONT (нажатием Ctrl-Q) . Проанализируйте поведение процесса и вывод на консоль, а также сравните с программой из предыдущего пункта.

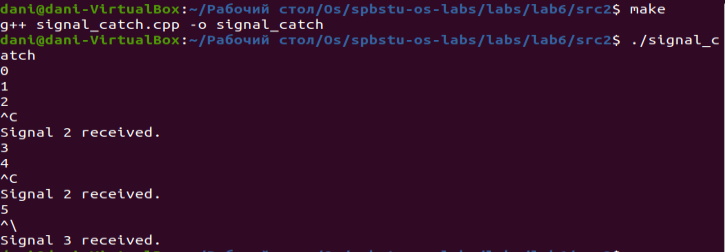


Рис. 6-2 Работа signal\_catch.cpp.

В отличии от программы из прошлого пункта производится обработка сигнала SIGQUIT.

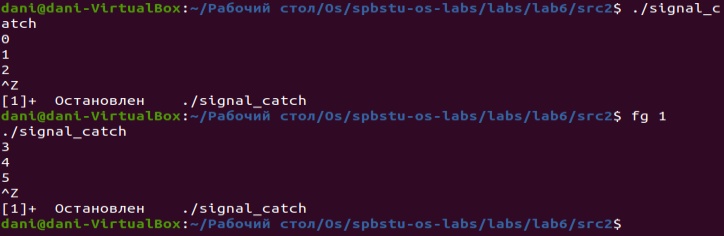


Рис. 6-3 SIGSTOP.

Сигнал SIGSTOP приостанавливает исполнение процесса и помещает его

в фоновый режим. Продолжить исполнение можно через команду **fg** или отправив сигнал SIGCONT (через команду **kill**).



Рис. 6-4 SIGCONT.

Однако процесс продолжит исполнение в фоновом режиме, из-за чего могут возникнуть проблемы с выводом на консоль.

**Пункт 3**

Скомпилируйте и запустите программу sigusr.cpp. Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой kill различные сигналы, в том числе и SIGUSR1 , проанализируйте реакцию на них.

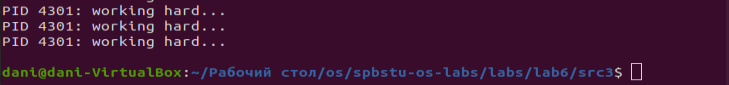




Рис. 6-5 Реакция на SIGINT.

Процесс завершился.





Рис. 6-6 Реакция на SIGQUIT.

Процесс завершаился с дампом памяти.



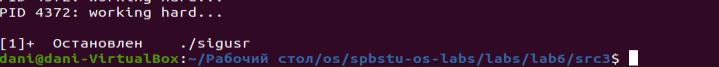


Рис. 6-7 Реакция на SIGSTOP.

Процесс приостанавливился и ушел в фоновый режим.

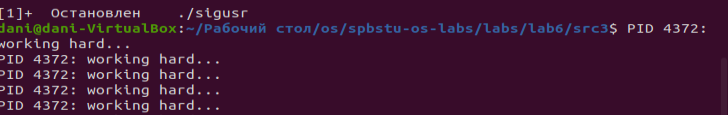


Рис. 6-8 Реакция на SIGCONT после SIGSTOP.

Процесс возобновил работу в фоновом режиме.

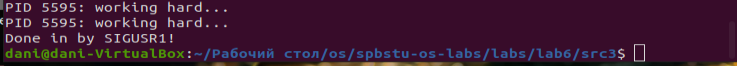




Рис. 6-9 Реакция на SIGCONT после SIGSTOP.

Процесс перехватил сигнал, вывел соотвествующее сообщение на консоль и завершился.

**Пункт 4**

Составьте программу, запускающую процесс-потомок. Процесс-родитель и процесс-потомок должны генерировать (можно случайным образом) и отправлять друг другу сигналы (например, SIGUSR1, SIGUSR2). Каждый из процессов должен выводить на консоль информацию об отправленном и о полученном сигналах.

Код составленной программы предостален ниже:

|  |
| --- |
| Листинг 6-1 signal\_exchange.cpp |
|  |

Итог выполнения программы:

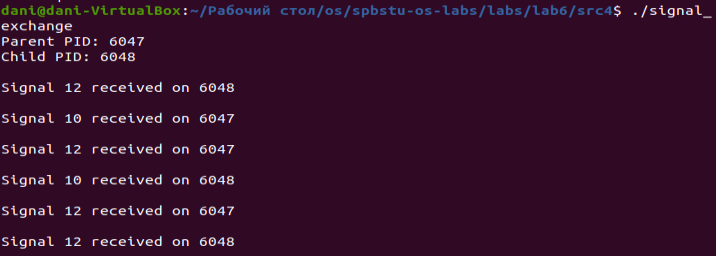


Рис. 6-10 Результат выполнения программы signal\_exchange.

**Пункт 5**

Для организации обработчиков сигналов предпочтительно использовать системный вызов sigaction() и соответствующую структуру данных. Обеспечьте корректное завершение процессов.

Для лучшей организации обработчиков использовался системный вызов sigaction():

|  |
| --- |
| Листинг 6-2 signal\_exchange\_v2.cpp |
|  |

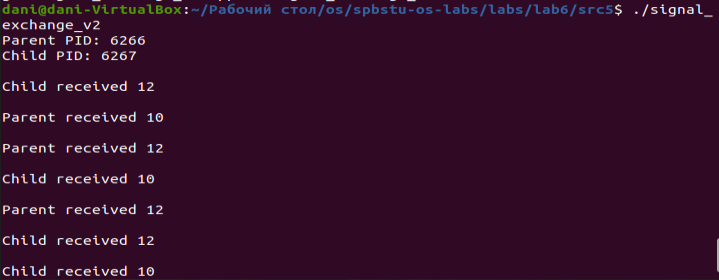


Рис. 6-11 Результат выполнения программы signal\_exchange\_v2.

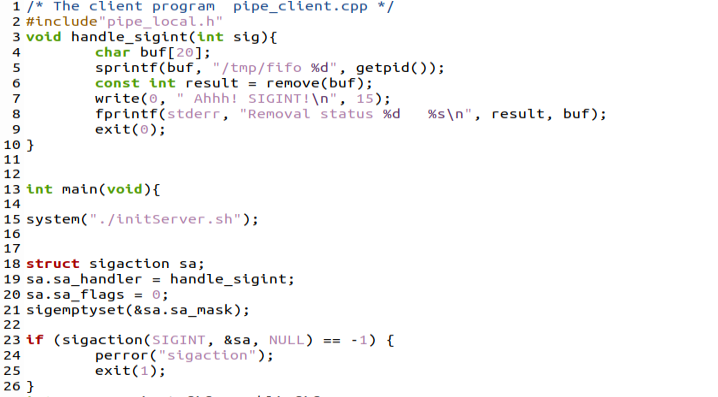
**Пункт 6**

Модифицируйте программу занятия 3 (файлы pipe\_server.cpp , pipe\_client.cpp и pipe\_local.h), сделав ее более стабильной вработе. В числе недостатков, которые желательно устранить, можно указать:

* если клиентский процесс завершается по получению сигнала SIGINT (Ctrl+C), то private FIFO не удаляется из системы (исправляется посредством организации перехвата сигнала с выполнением необходимых действий);
* клиентский процесс при его инициализации может обрушиться, если сервер окажется недоступен (исправляется путем попытки запуска сервера из клиента, если сервер не активен).

Внесем изменения:

* При завершении процесса сервера сигналом SIGINT сервер удаляет PUBLIC fifo
* При завершении процесса клиента сигналом SIGINT клиент удаляет private fifo
* При отсутствии процесса сервера на момент запуска процесса клиента, сервер запускается автоматически

Рис. 6-12 Модификации в файле pipe\_client.cpp.

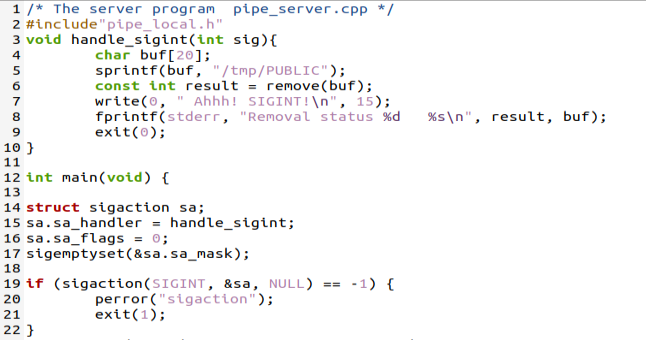


Рис. 6-13 Модификации в файле pipe\_server.cpp.



Рис. 6-14 Содержимое файла initServer.sh

Sleep 1 нам нужен для того чтобы сервер успел запуститься, иначе будет ошибка о том что соответсвубщее fifo не найдено.

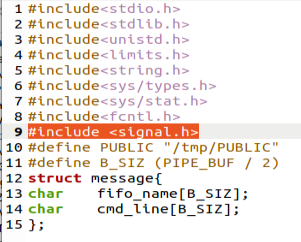


Рис. 6-13 Модификация в файле pipe\_local.h.

Проверим работоспособность.

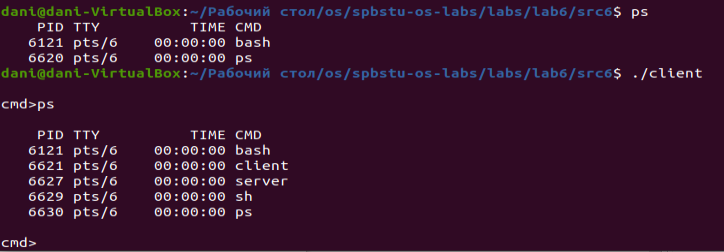


Рис. 6-14 Запуск клиента без сервера.

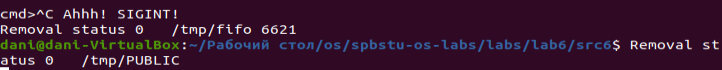


Рис. 6-15 Обработка SIGINT.

Если закрыть клиент через SIGINT он закроет свой private FIFO.

Причем если сервер был запущен через клиент, сервер завершит свою работу по завершению работы клиента, иначе сервер останется работать.

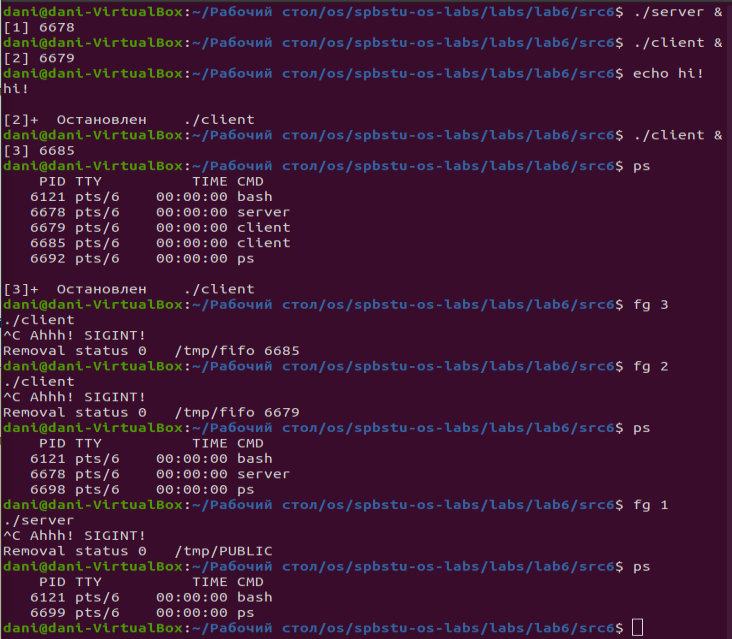


Рис. 6-16 Штатный режим работы.

Вот что происходило в дирректории tmp в этот момент (Рис. 6-16):

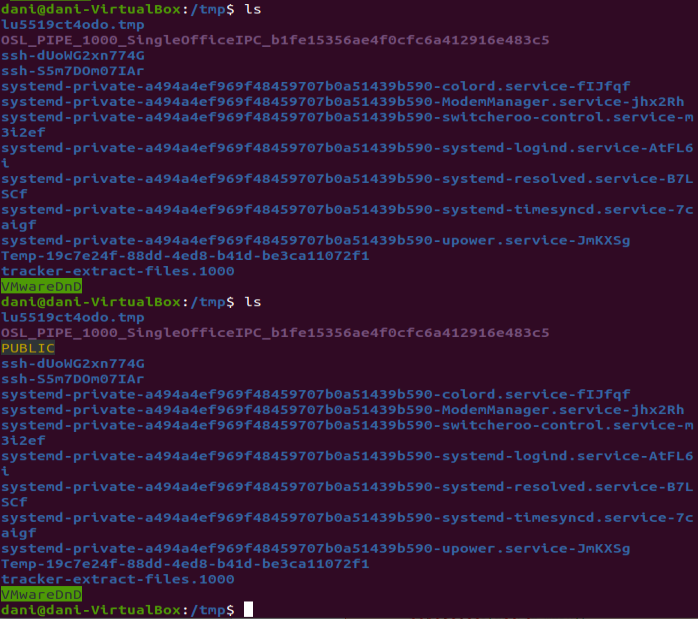


Рис. 6-17 Содержимое tmp до создания сервера.

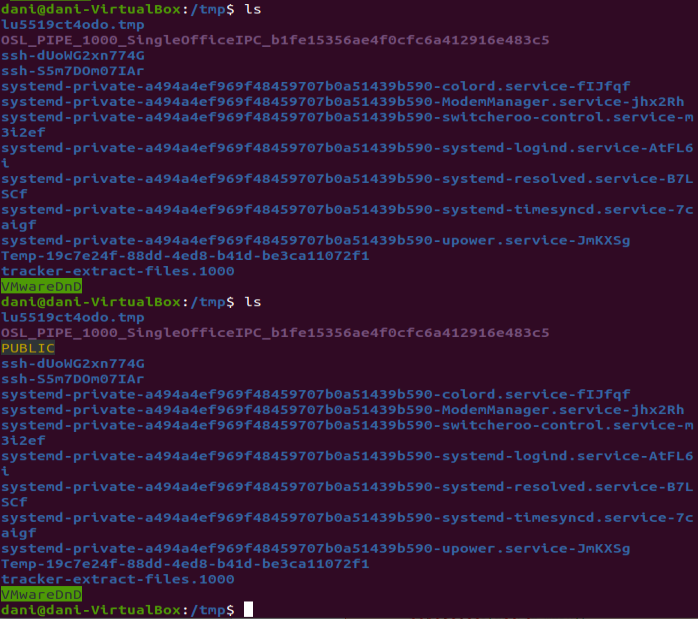


Рис. 6-18 Содержимое tmp после создания сервера.

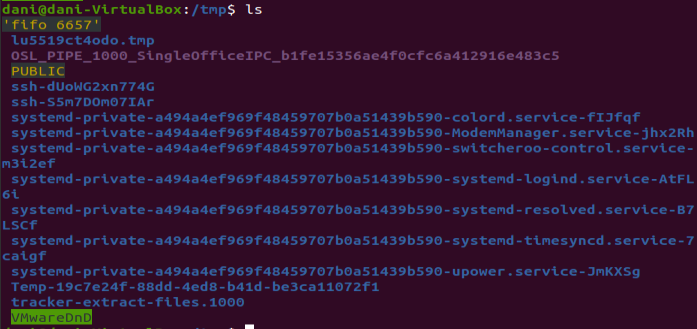


Рис. 6-19 Содержимое tmp после создания 1 клиента.



Рис. 6-20 Содержимое tmp после создания 2 клиента.

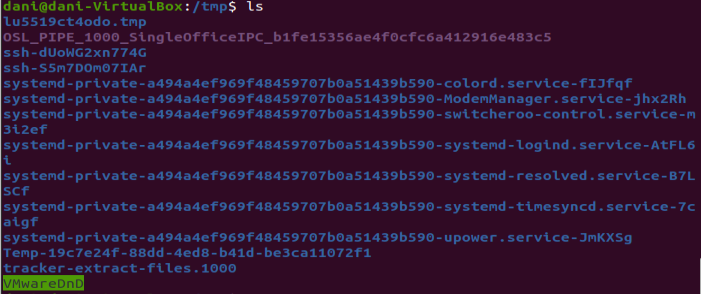


Рис. 6-21 Содержимое tmp в итоге.

Все полученные команды исполняются в дочерних процессах.

**Вывод**

В лабораторной работе был рассмотрен способ взаимодействия с процессами через различные сигналы, а также мы познакомились генерацией сигналов в операционной системе Linux.