

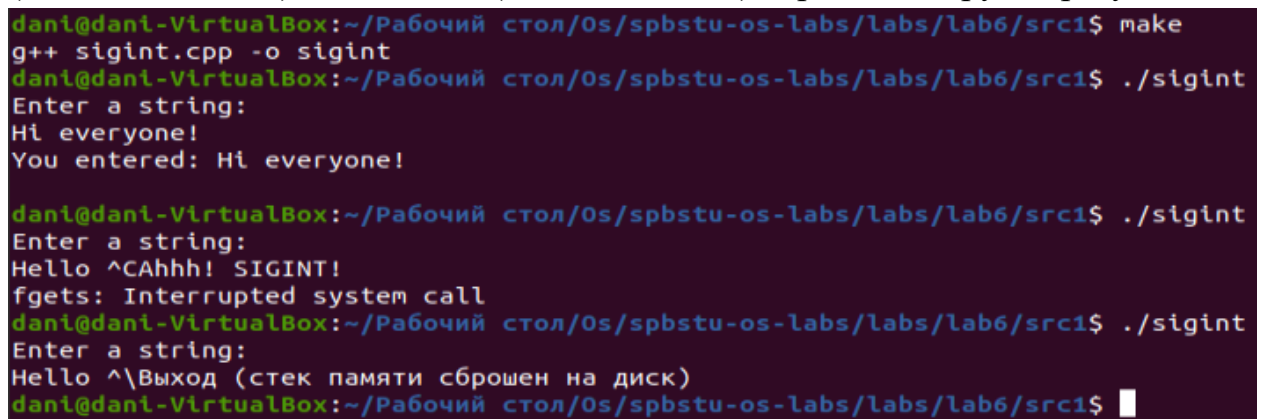
Лабораторная работа № 6 – Генерация и обработка сигналов

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 (нажатием Ctrl-\\). Проанализируйте результаты.



```
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src1$ make
g++ sigint.cpp -o sigint
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src1$ ./sigint
Enter a string:
Hi everyone!
You entered: Hi everyone!

dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src1$ ./sigint
Enter a string:
Hello ^CAhhh! SIGINT!
fgets: Interrupted system call
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src1$ ./sigint
Enter a string:
Hello ^\\Выход (стек памяти сброшен на диск)
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src1$
```

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.

```

dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ make
g++ signal_catch.cpp -o signal_catch
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ ./signal_c
atch
0
1
2
^C
Signal 2 received.
3
4
^C
Signal 2 received.
5
^\\
Signal 3 received.

```

Рис. 6-2 Работа signal_catch.cpp.

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

```

dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ ./signal_c
atch
0
1
2
^Z
[1]+  Остановлен    ./signal_catch
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ fg 1
./signal_catch
3
4
5
^Z
[1]+  Остановлен    ./signal_catch
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$

```

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**).

```

dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ ./signal_c
atch
0
1
2
^Z
[1]+  Остановлен    ./signal_catch
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ ps
  PID TTY          TIME CMD
 23464 pts/0      00:00:00 bash
 23487 pts/0      00:00:00 signal_catch
 23492 pts/0      00:00:00 ps
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ kill -SIGC
ONT 23487
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ 3
4
5
6
7
^C
dani@dani-VirtualBox:~/Рабочий стол/0s/spbstu-os-labs/labs/lab6/src2$ 8
9

```

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 закидывается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой kill различные сигналы, в том числе и SIGUSR1, проанализируйте реакцию на них.

```
PID 4301: working hard...
PID 4301: working hard...

dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src3$ 
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$ kill -SIGINT 4301
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$
```

Рис. 6-5 Реакция на SIGINT.

Процесс завершился.

```
PID 4361: working hard...
Выход (стек памяти сброшен на диск)
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src3$ 
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$ kill -SIGQUIT 4361
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$
```

Рис. 6-6 Реакция на SIGQUIT.

Процесс завершался с дампом памяти.

```
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$ kill -SIGSTOP 4372
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$

PID 4372: working hard...

[1]+  Остановлен    ./sigusr
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src3$
```

Рис. 6-7 Реакция на SIGSTOP.

Процесс приостановлился и ушел в фоновый режим.

```
[1]+  Остановлен    ./sigusr
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src3$ PID 4372:
working hard...
PID 4372: working hard...
PID 4372: working hard...
```

Рис. 6-8 Реакция на SIGCONT после SIGSTOP.

Процесс возобновил работу в фоновом режиме.

```
PID 5595: working hard...
PID 5595: working hard...
Done in by SIGUSR1!
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src3$ 
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$ kill -SIGUSR1 5595
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs$
```

Рис. 6-9 Реакция на SIGCONT после SIGSTOP.

Процесс перехватил сигнал, вывел соответствующее 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 предоставлен ниже:

Листинг 6-1 signal_exchange.cpp

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5 #include <sys/types.h>
6
7 main(void)
8 {
9     void signal1(int);
10
11     if (signal(SIGUSR1, signal1) == SIG_ERR)
12     {
13         exit(1);
14     }
15
16     if (signal(SIGUSR2, signal1) == SIG_ERR)
17     {
18         exit(2);
19     }
20
21     int p_pid = getpid();
22     int c_pid = fork();
23     if (c_pid == -1)
24     {
25         perror("Bad fork");
26         exit(4);
27     }
28
29     if (c_pid == 0)
30     {
31         c_pid = getpid();
32         printf("Child PID: %d\n\n", c_pid);
33
34         for (int i = 1;; i++)
35         {
36             sleep(1);
37             switch (i % 3)
38             {
39                 case 0:
40                     kill(c_pid, SIGUSR1);
41                     break;
42                 case 1:
43                     kill(p_pid, SIGUSR1);
44                     break;
45                 default:
46                     kill(p_pid, SIGUSR2);
47             }
48         }
49     }
50     else
51     {
52         printf("Parent PID: %d\n", p_pid);
53
54         for (int i = 1;; i++)
55         {
56             sleep(1);
57             switch (i % 3)
58             {
59                 case 0:
60                     kill(c_pid, SIGUSR1);
61                     break;
62                 case 1:
63                     kill(c_pid, SIGUSR2);
64                     break;
65                 default:
66                     kill(p_pid, SIGUSR2);
67             }
68         }
69     }
70 }
71
72 void signal1(int the_sig)
73 {
74     signal(the_sig, signal1);
75     printf("Signal %d received on %d\n\n", the_sig, getpid());
76 }
77 }
```

Итог выполнения программы:

```

dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src4$ ./signal_
exchange
Parent PID: 6047
Child PID: 6048

Signal 12 received on 6048
Signal 10 received on 6047
Signal 12 received on 6047
Signal 10 received on 6048
Signal 12 received on 6047
Signal 12 received on 6048

```

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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <errno.h>
5 #include <signal.h>
6
7 int main(void) {
8
9     void signal_p(int);
10    void signal_c(int);
11    struct sigaction sa;
12    sa.sa_flags = 0;
13    sigemptyset(&sa.sa_mask);
14    int p_pid = getpid();
15    int c_pid = fork();
16    if (c_pid == -1) {
17        perror("Bad fork");
18        exit(4);
19    }
20
21    if (c_pid == 0) {
22
23        c_pid = getpid();
24        printf("Child PID: %d\n\n", c_pid);
25
26        sa.sa_handler = signal_c;
27        if (sigaction(SIGUSR1, &sa, NULL) == -1) {
28            printf("Signal not received");
29            perror("SIGQUIT");
30            exit(1);
31        }
32        if (sigaction(SIGUSR2, &sa, NULL) == -1) {
33            printf("Signal not received");
34            perror("SIGQUIT");
35            exit(1);
36        }
37
38        for (int i = 1;; i++) {
39            sleep(1);
40            switch (i % 3) {
41                case 0:
42                    kill(c_pid, SIGUSR1);
43                    break;
44                case 1:
45                    kill(p_pid, SIGUSR1);
46                    break;
47                default:
48                    kill(p_pid, SIGUSR2);
49            }

```

```

50     }
51 }
52 } else {
53     printf("Parent PID: %d\n", p_pid);
54     sa.sa_handler = signal_p;
55     if (sigaction(SIGUSR1, &sa, NULL) == -1) {
56         printf("Signal not received");
57         perror("SIGQUIT");
58         exit(1);
59     }
60     if (sigaction(SIGUSR2, &sa, NULL) == -1) {
61         printf("Signal not received");
62         perror("SIGQUIT");
63         exit(1);
64     }
65     for (int i = 1;; i++) {
66         sleep(1);
67         switch (i % 3) {
68             case 0:
69                 kill(c_pid, SIGUSR1);
70                 break;
71             case 1:
72                 kill(c_pid, SIGUSR2);
73                 break;
74             default:
75                 kill(p_pid, SIGUSR2);
76         }
77     }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 void signal_p(int the_sig) {
89     signal(the_sig, signal_p);
90     printf("Parent received %d\n\n", the_sig);
91     if (the_sig == SIGQUIT) {
92         exit(1);
93     }
94 }
95 }
96 void signal_c(int the_sig) {
97     signal(the_sig, signal_c);
98     printf("Child received %d\n\n", the_sig);
99     if (the_sig == SIGQUIT) {
100         exit(1);
101     }
102 }

```

```

dant@dant-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src5$ ./signal_
exchange_v2
Parent PID: 6266
Child PID: 6267

Child received 12
Parent received 10
Parent received 12
Child received 10
Parent received 12
Child received 12
Child received 10

```

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 в работе. 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 обраться, 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

```
1 /* The client program pipe_client.cpp */
2 #include "pipe_local.h"
3 void handle_sigint(int sig){
4     char buf[20];
5     sprintf(buf, "/tmp/fifo %d", getpid());
6     const int result = remove(buf);
7     write(0, " Ahhh! SIGINT!\n", 15);
8     fprintf(stderr, "Removal status %d   %s\n", result, buf);
9     exit(0);
10 }
11
12
13 int main(void){
14
15     system("./initServer.sh");
16
17
18     struct sigaction sa;
19     sa.sa_handler = handle_sigint;
20     sa.sa_flags = 0;
21     sigemptyset(&sa.sa_mask);
22
23     if (sigaction(SIGINT, &sa, NULL) == -1) {
24         perror("sigaction");
25         exit(1);
26 }
```

Рис. 6-12 Модификации в файле `pipe_client.cpp`.

```

1 /* The server program pipe_server.cpp */
2 #include "pipe_local.h"
3 void handle_sigint(int sig){
4     char buf[20];
5     sprintf(buf, "/tmp/PUBLIC");
6     const int result = remove(buf);
7     write(0, " Ahhh! SIGINT!\n", 15);
8     fprintf(stderr, "Removal status %d   %s\n", result, buf);
9     exit(0);
10 }
11
12 int main(void) {
13
14     struct sigaction sa;
15     sa.sa_handler = handle_sigint;
16     sa.sa_flags = 0;
17     sigemptyset(&sa.sa_mask);
18
19     if (sigaction(SIGINT, &sa, NULL) == -1) {
20         perror("sigaction");
21         exit(1);
22     }

```

Рис. 6-13 Модификации в файле pipe_server.cpp.

```

1 #!/bin/bash
2
3 if [[ "`ps | grep server`" == "" ]]
4 then
5     ./server &
6     sleep 1
7 fi

```

Рис. 6-14 Содержимое файла initServer.sh

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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <limits.h>
5 #include <string.h>
6 #include <sys/types.h>
7 #include <sys/stat.h>
8 #include <fcntl.h>
9 #include <signal.h>
10 #define PUBLIC "/tmp/PUBLIC"
11 #define B_SIZ (PIPE_BUF / 2)
12 struct message{
13     char    fifo_name[B_SIZ];
14     char    cmd_line[B_SIZ];
15 };

```

Рис. 6-13 Модификация в файле pipe_local.h.

Проверим работоспособность.


```

dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ps
  PID TTY          TIME CMD
  6121 pts/6        00:00:00 bash
  6620 pts/6        00:00:00 ps
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ./client
cmd>ps

  PID TTY          TIME CMD
  6121 pts/6        00:00:00 bash
  6621 pts/6        00:00:00 client
  6627 pts/6        00:00:00 server
  6629 pts/6        00:00:00 sh
  6630 pts/6        00:00:00 ps

```

Рис. 6-14 Запуск клиента без сервера.

```

cmd>^C Ahhh! SIGINT!
Removal status 0 /tmp/fifo 6621
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ Removal status 0 /tmp/PUBLIC

```

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.

```

dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ./server &
[1] 6678
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ./client &
[2] 6679
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ echo hi!
hi!

[2]+  Остановлен ./client
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ./client &
[3] 6685
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ps
  PID TTY          TIME CMD
  6121 pts/6        00:00:00 bash
  6678 pts/6        00:00:00 server
  6679 pts/6        00:00:00 client
  6685 pts/6        00:00:00 client
  6692 pts/6        00:00:00 ps

[3]+  Остановлен ./client
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ fg 3
./client
^C Ahhh! SIGINT!
Removal status 0 /tmp/fifo 6685
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ fg 2
./client
^C Ahhh! SIGINT!
Removal status 0 /tmp/fifo 6679
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ps
  PID TTY          TIME CMD
  6121 pts/6        00:00:00 bash
  6678 pts/6        00:00:00 server
  6698 pts/6        00:00:00 ps
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ fg 1
./server
^C Ahhh! SIGINT!
Removal status 0 /tmp/PUBLIC
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ ps
  PID TTY          TIME CMD
  6121 pts/6        00:00:00 bash
  6699 pts/6        00:00:00 ps
dani@dani-VirtualBox:~/Рабочий стол/os/spbstu-os-labs/labs/lab6/src6$ 

```

Рис. 6-16 Штатный режим работы.

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

```
dani@dani-VirtualBox:/tmp$ ls
lu5519ct4odo.tmp
OSL_PIPE_1000_SingleOfficeIPC_b1fe15356ae4f0cfc6a412916e483c5
ssh-dUoWG2xn774G
ssh-S5m7D0m07IAR
systemd-private-a494a4ef969f48459707b0a51439b590-colord.service-fIJfqf
systemd-private-a494a4ef969f48459707b0a51439b590-ModemManager.service-jhx2Rh
systemd-private-a494a4ef969f48459707b0a51439b590-switcheroo-control.service-m
3i2ef
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-logind.service-AtFL6
i
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-resolved.service-B7L
SCf
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-timesyncd.service-7c
aigf
systemd-private-a494a4ef969f48459707b0a51439b590-upower.service-JmKXSg
Temp-19c7e24f-88dd-4ed8-b41d-be3ca11072f1
tracker-extract-files.1000
VMwareDnD
```

Рис. 6-17 Содержимое tmp до создания сервера.

```
dani@dani-VirtualBox:/tmp$ ls
lu5519ct4odo.tmp
OSL_PIPE_1000_SingleOfficeIPC_b1fe15356ae4f0cfc6a412916e483c5
PUBLIC
ssh-dUoWG2xn774G
ssh-S5m7D0m07IAR
systemd-private-a494a4ef969f48459707b0a51439b590-colord.service-fIJfqf
systemd-private-a494a4ef969f48459707b0a51439b590-ModemManager.service-jhx2Rh
systemd-private-a494a4ef969f48459707b0a51439b590-switcheroo-control.service-m
3i2ef
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-logind.service-AtFL6
i
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-resolved.service-B7L
SCf
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-timesyncd.service-7c
aigf
systemd-private-a494a4ef969f48459707b0a51439b590-upower.service-JmKXSg
Temp-19c7e24f-88dd-4ed8-b41d-be3ca11072f1
tracker-extract-files.1000
VMwareDnD
```

Рис. 6-18 Содержимое tmp после создания сервера.

```
dani@dani-VirtualBox:/tmp$ ls
'fifo 6657'
lu5519ct4odo.tmp
OSL_PIPE_1000_SingleOfficeIPC_b1fe15356ae4f0cfc6a412916e483c5
PUBLIC
ssh-dUoWG2xn774G
ssh-S5m7D0m07IAR
systemd-private-a494a4ef969f48459707b0a51439b590-colord.service-fIJfqf
systemd-private-a494a4ef969f48459707b0a51439b590-ModemManager.service-jhx2Rh
systemd-private-a494a4ef969f48459707b0a51439b590-switcheroo-control.service-
m3i2ef
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-logind.service-AtFL
6i
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-resolved.service-B7
LSCf
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-timesyncd.service-7
caigf
systemd-private-a494a4ef969f48459707b0a51439b590-upower.service-JmKXSg
Temp-19c7e24f-88dd-4ed8-b41d-be3ca11072f1
tracker-extract-files.1000
VMwareDnD
```

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

```

dani@dani-VirtualBox:/tmp$ ls
'fifo 6657'
'fifo 6665'
lu5519ct4odo.tmp
OSL_PIPE_1000_SingleOfficeIPC_b1fe15356ae4f0cfc6a412916e483c5
PUBLIC
ssh-dUoWG2xn774G
ssh-S5m7D0m07IAR
systemd-private-a494a4ef969f48459707b0a51439b590-colord.service-fIJfqf
systemd-private-a494a4ef969f48459707b0a51439b590-ModemManager.service-jhx2Rh
systemd-private-a494a4ef969f48459707b0a51439b590-switcheroo-control.service-m3i2ef
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-logind.service-AtFL6i
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-resolved.service-B7LSCf
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-timesyncd.service-7caigf
systemd-private-a494a4ef969f48459707b0a51439b590-upower.service-JmKXSg
Temp-19c7e24f-88dd-4ed8-b41d-be3ca11072f1
tracker-extract-files.1000
VMwareDnD

```

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

```

dani@dani-VirtualBox:/tmp$ ls
lu5519ct4odo.tmp
OSL_PIPE_1000_SingleOfficeIPC_b1fe15356ae4f0cfc6a412916e483c5
ssh-dUoWG2xn774G
ssh-S5m7D0m07IAR
systemd-private-a494a4ef969f48459707b0a51439b590-colord.service-fIJfqf
systemd-private-a494a4ef969f48459707b0a51439b590-ModemManager.service-jhx2Rh
systemd-private-a494a4ef969f48459707b0a51439b590-switcheroo-control.service-m3i2ef
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-logind.service-AtFL6i
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-resolved.service-B7LSCf
systemd-private-a494a4ef969f48459707b0a51439b590-systemd-timesyncd.service-7caigf
systemd-private-a494a4ef969f48459707b0a51439b590-upower.service-JmKXSg
Temp-19c7e24f-88dd-4ed8-b41d-be3ca11072f1
tracker-extract-files.1000
VMwareDnD

```

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.