



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №4 по курсу "Операционные системы"

Тема Процессы. Системные вызовы fork и exec

Студент Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Задание 1

Процессы-сироты. В программе создаются не менее двух потомков. В потомках вызывается `sleep()`. Чтобы предок гарантированно завершился раньше своих потомков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе. Продемонстрировать «усыновление». Для этого надо в потомках вывести идентификаторы: собственный, предка, группы до блокировки и после блокировки.

На листинге 1 представлен код программы:

Листинг 1: Код задания 1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SLEEP_TIME 3

int children_pids[2];

int main(void)
{
    printf("Parent process: PID=%d, GROUP=%d\n", getpid(), getpgrp());

    for (size_t i = 0; i < 2; i++)
    {
        int pid;
        if ((pid = fork()) == -1)
        {
            perror("Can't fork\n");
            return EXIT_FAILURE;
        }
        else if (pid == 0)
        {
            printf("\nChild process before kill : PID=%d, GROUP=%d, PPID=%d\n",
                getpid(), getpgrp(), getppid());
            sleep(SLEEP_TIME);
            printf("\nChild process after kill : PID=%d, GROUP=%d, PPID=%d\n",
                getpid(), getpgrp(), getppid());
            return EXIT_SUCCESS;
        }
        else
        {

```

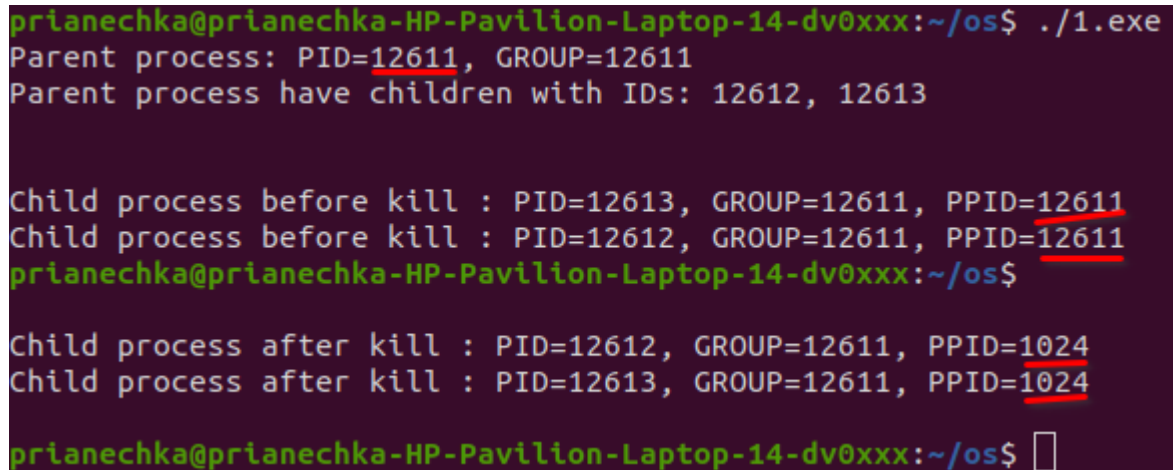
```

        children_pids[i] = pid;
    }
}
printf("Parent process have children with IDs: %d, %d\n",
       children_pids[0], children_pids[1]);

return EXIT_SUCCESS;
}

```

На рисунке 1 продемонстрирован вывод программы:



```

prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./1.exe
Parent process: PID=12611, GROUP=12611
Parent process have children with IDs: 12612, 12613

Child process before kill : PID=12613, GROUP=12611, PPID=12611
Child process before kill : PID=12612, GROUP=12611, PPID=12611
prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$

Child process after kill : PID=12612, GROUP=12611, PPID=1024
Child process after kill : PID=12613, GROUP=12611, PPID=1024
prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ 

```

Рисунок 1: Демонстрация работы программы

Лабораторная выполнялась на Ubuntu. Как видно, идентификатор предка для потомков действительно сменился на идентификатора процесса-посредника.

Задание 2

Предок ждет завершения своих потомком, используя системный вызов `wait()`. Вывод соответствующих сообщений на экран. В программе необходимо, чтобы предок выполнял анализ кодов завершения потомков.

На листинге 2 представлен код программы:

Листинг 2: Код задания 2

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define SLEEP_TIME 3

int children_pids[2];

int main(void)
{
    printf("Parent process: PID=%d, GROUP=%d\n", getpid(), getpgrp());

    for (size_t i = 0; i < 2; i++)
    {
        int pid;
        if ((pid = fork()) == -1)
        {
            perror("Can't fork\n");
            return EXIT_FAILURE;
        }
        else if (pid == 0)
        {
            sleep(SLEEP_TIME);
            printf("\nChild process : PID=%d, GROUP=%d, PPID=%d\n", getpid(), getpgrp(), getppid());
            return EXIT_SUCCESS;
        }
        else
        {
            children_pids[i] = pid;
        }
    }

    for (size_t i = 0; i < 2; i++)
    {
        int status;
        pid_t childpid = wait(&status);
```

```

printf("\n\nChild process finished: PID = %d, status = %d\n",
      childpid, status);

if (WIFEXITED(status))
{
    printf("Дочерний процесс завершён корректно.\n");
    printf("Child process exited with code %d\n", WEXITSTATUS(
        status));
}
else if (WIFSIGNALED(status))
{
    printf("Дочерний процесс завершён неперехватываемым сигналом\n
        ");
    printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));
}
else if (WIFSTOPPED(status))
{
    printf ("Дочерний процесс остановлен\n");
    printf ("Номер сигнала: \t%d\n\n", WSTOPSIG (status));
}
}

return EXIT_SUCCESS;
}

```

На рисунке 2 продемонстрирован вывод программы:

```

prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./wait.exe
Parent process: PID=13585, GROUP=13585

Child process : PID=13586, GROUP=13585, PPID=13585
Child process : PID=13587, GROUP=13585, PPID=13585

Child process finished: PID = 13586, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0

Child process finished: PID = 13587, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0
prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ 

```

Рисунок 2: Демонстрация работы программы

Видно, что сначала завершились все потомки, а только затем – предок.

Задание 3

Потомки переходят на выполнение других программ, которые передаются системному вызову `exec()` в качестве параметра. Потомки должны выполнять разные программы. Предок ждет завершения своих потомков с анализом кодов завершения. На экран выводятся соответствующие сообщения.

На листинге 3 представлен код программы:

Листинг 3: Код задания 3

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define OK 0
#define EXEC_ERROR -1
#define ERROR -1
#define SLEEP_TIME 3

int children_pids[2];

int main(void)
{
    printf("Parent process: PID=%d, GROUP=%d\n", getpid(), getpgrp());

    int pid;
    if ((pid = fork()) == -1)
    {
        perror("Can't fork\n");
        return EXIT_FAILURE;
    }
    else if (pid == 0)
    {
        printf("\nChild process : PID=%d, GROUP=%d, PPID=%d\n", getpid(),
            getpgrp(), getppid());
        if (execlp("./qck/quicksort.exe", "quicksort.exe", NULL) ==
            EXEC_ERROR)
        {
            printf ("\nError : Child 1 can not execute exec ()\n");
            exit(ERROR);
        }
        exit(OK);
    }
    else
```

```

{
    children_pids[0] = pid;
}

if ((pid = fork()) == -1)
{
    perror("Can't fork\n");
    return EXIT_FAILURE;
}
else if (pid == 0)
{
    printf("\nChild process : PID=%d, GROUP=%d, PPID=%d\n\n", getpid()
        , getpgrp(), getppid());
    if (execlp("./aa/app.exe", "app.exe", NULL) == EXEC_ERROR)
    {
        printf("\nError : Child 2 can not execute exec ()\n");
        exit(ERROR);
    }
    exit(OK);
}
else
{
    children_pids[1] = pid;
}

for (size_t i = 0; i < 2; i++)
{
    int status;
    pid_t childpid = wait(&status);
    printf("\n\nChild process finished: PID = %d, status = %d\n",
        childpid, status);

    if (WIFEXITED(status))
    {
        printf("Child process exited with code %d\n\n\n", WEXITSTATUS(
            status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершен перехватываемым сигналом\n
        ");
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));
    }
    else if (WIFSTOPPED(status))
    {
        printf("Дочерний процесс остановлен\n");
        printf("Номер сигнала: \t%d\n\n", WSTOPSIG(status));
    }
}

```

```
        }  
    }  
  
    return EXIT_SUCCESS;  
}
```

В качестве программ, которые вызываются программой exes выбраны:

1. Быстрая сортировка массива;
2. Программа вычисления наиболее коррелирующих столбцов матрицы при помощи поточных вычислений (из курса "Анализ Алгоритмов")

На рисунке 3 продемонстрирован вывод программы:


```
prlanechka@prlanechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./3.exe
Parent process: PID=19695, GROUP=19695

Child process : PID=19696, GROUP=19695, PPID=19695

Child process : PID=19697, GROUP=19695, PPID=19695

Исходный массив:
1 5 23 -4 6 3 0 -5 9 11

После быстрой сортировки:
-5 -4 0 1 3 5 6 9 11 23

Child process finished: PID = 19696, status = 0
Child process exited with code 0

Исходная матрица:
5 9 1 9 2
1 3 8 5 0
5 5 6 0 6
7 8 0 1 5
8 9 1 7 6
2 3 3 3 9

Результаты работы алгоритма с использованием потоков:
Индексы наиболее коррелирующих столбцов: 0, 1
Значение корреляции: 0.879347

Child process finished: PID = 19697, status = 0
Child process exited with code 0

prlanechka@prlanechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$
```

Рисунок 3: Демонстрация работы программы

Также в этом задании была смоделирована ситуация, когда дочерний процесс не был корректно завершён:

```
prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./3.exe
Parent process: PID=30185, GROUP=30185

Child process : PID=30186, GROUP=30185, PPID=30185

Child process : PID=30187, GROUP=30185, PPID=30185

Исходный массив:
1 5 23 -4 6 3 0 -5 9 11

После быстрой сортировки:
-5 -4 0 1 3 5 6 9 11 23

Child process finished: PID = 30186, status = 0
Child process exited with code 0

terminate called after throwing an instance of 'std::bad_alloc'
what():  std::bad_alloc

Child process finished: PID = 30187, status = 134
Дочерний процесс завершен перехватываемым сигналом
Номер сигнала:  6

prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$
```

Рисунок 4: Демонстрация работы программы

Задание 4

Предок и потомки обмениваются сообщениями через неименованный программный канал. Причем оба потомка пишут свои сообщения в один программный канал, а предок их считывает из канала. Потомки должны посылать предку разные сообщения по содержанию и размеру. Предок считывает сообщения от потомков и выводит их на экран. Предок ждет завершения своих потомков и анализирует код их завершения. Вывод соответствующих сообщений на экран.

На листинге 4 представлен код программы:

Листинг 4: Код задания 4

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define SLEEP_TIME 3

int children_pids[2];
const char volatile *messages[] = {"\n1. Первое сообщение\n", "2. SecMes"
    };

int main(void)
{
    int fd[2];
    char buf[50] = {0};
    printf("Parent process: PID=%d, GROUP=%d\n", getpid(), getpgrp());

    if (pipe(fd) == -1)
    {
        perror("Can't pipe\n");
        return EXIT_FAILURE;
    }

    for (size_t i = 0; i < 2; i++)
    {
        int pid;
        if ((pid = fork()) == -1)
        {
            perror("Can't fork\n");
            return EXIT_FAILURE;
        }
    }
```

```

else if (pid == 0)
{
    printf("\nChild process : PID=%d, GROUP=%d, PPID=%d\n", getpid
        (), getpgrp(), getppid());
    close(fd[0]);
    write(fd[1], messages[i], strlen(messages[i]));
    printf("Message from child has been sent to parent\n");
    exit(EXIT_SUCCESS);
}
else
{
    children_pids[i] = pid;
}
}

for (size_t i = 0; i < 2; i++)
{
    int status;
    pid_t childpid = wait(&status);
    printf("\n\nChild process finished: PID = %d, status = %d\n",
        childpid, status);

    if (WIFEXITED(status))
    {
        printf("Дочерний процесс завершён корректно.\n");
        printf("Child process exited with code %d\n", WEXITSTATUS(
            status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершён неперехватываемым сигналом\n
            ");
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));
    }
    else if (WIFSTOPPED(status))
    {
        printf ("Дочерний процесс остановлен\n");
        printf ("Номер сигнала: \t%d\n\n", WSTOPSIG (status));
    }
}

close(fd[1]);
read(fd[0], buf, sizeof(buf));
printf("\nReceived messages: %s\n", buf);

return EXIT_SUCCESS;
}

```

На рисунке 5 продемонстрирован вывод программы:

```
prianechka@prianechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./4.exe
Parent process: PID=26481, GROUP=26481

Child process : PID=26482, GROUP=26481, PPID=26481
Message from child has been sent to parent

Child process : PID=26483, GROUP=26481, PPID=26481
Message from child has been sent to parent


Child process finished: PID = 26482, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0


Child process finished: PID = 26483, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0


Received messages:
1. Первое сообщение
2. SecMes
prianechka@prianechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$
```

Рисунок 5: Демонстрация работы программы

Задание 5

Предок и потомки аналогично п.4 обмениваются сообщениями через неименованный программный канал. В программу включается собственный обработчик сигнала. С помощью сигнала меняется ход выполнения программы. При получении сигнала потомки записывают сообщения в канал, если сигнал не поступает, то не записывают. Предок ждет завершения своих потомков и анализирует коды их завершений. Вывод соответствующих сообщений на экран.

На листинге 5 представлен код программы:

Листинг 5: Код задания 5

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

#define SLEEP_TIME 3
#define TRUE 1
#define FALSE 0

int children_pids[2];
const char volatile *messages[] = {"\n1. Первое сообщение\n", "2. SecMes"
    };
int flag = FALSE;

void catch_signal(int signal)
{
    printf("\nCaught signal: %d\n", signal);
    flag = TRUE;
}

int main(void)
{
    int fd[2];
    char buf[50] = {0};
    printf("Parent process: PID=%d, GROUP=%d\n", getpid(), getpgrp());

    signal(SIGINT, catch_signal);
    sleep(SLEEP_TIME);

    if (pipe(fd) == -1)
```

```

{
    perror("Can't pipe\n");
    return EXIT_FAILURE;
}

for (size_t i = 0; i < 2; i++)
{
    int pid;
    if ((pid = fork()) == -1)
    {
        perror("Can't fork\n");
        return EXIT_FAILURE;
    }
    else if (pid == 0)
    {
        printf("\nChild process : PID=%d, GROUP=%d, PPID=%d\n", getpid(
            ), getpgrp(), getppid());
        if (flag == TRUE)
        {
            close(fd[0]);
            write(fd[1], messages[i], strlen(messages[i]));
            printf("Message from child has been sent to parent\n");
        }
        exit(EXIT_SUCCESS);
    }
    else
    {
        children_pids[i] = pid;
    }
}

for (size_t i = 0; i < 2; i++)
{
    int status;
    pid_t childpid = wait(&status);
    printf("\n\nChild process finished: PID = %d, status = %d\n",
        childpid, status);

    if (WIFEXITED(status))
    {
        printf("Дочерний процесс завершён корректно.\n");
        printf("Child process exited with code %d\n", WEXITSTATUS(
            status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Дочерний процесс завершён неперехватываемым сигналом\n
            ");
    }
}

```

```

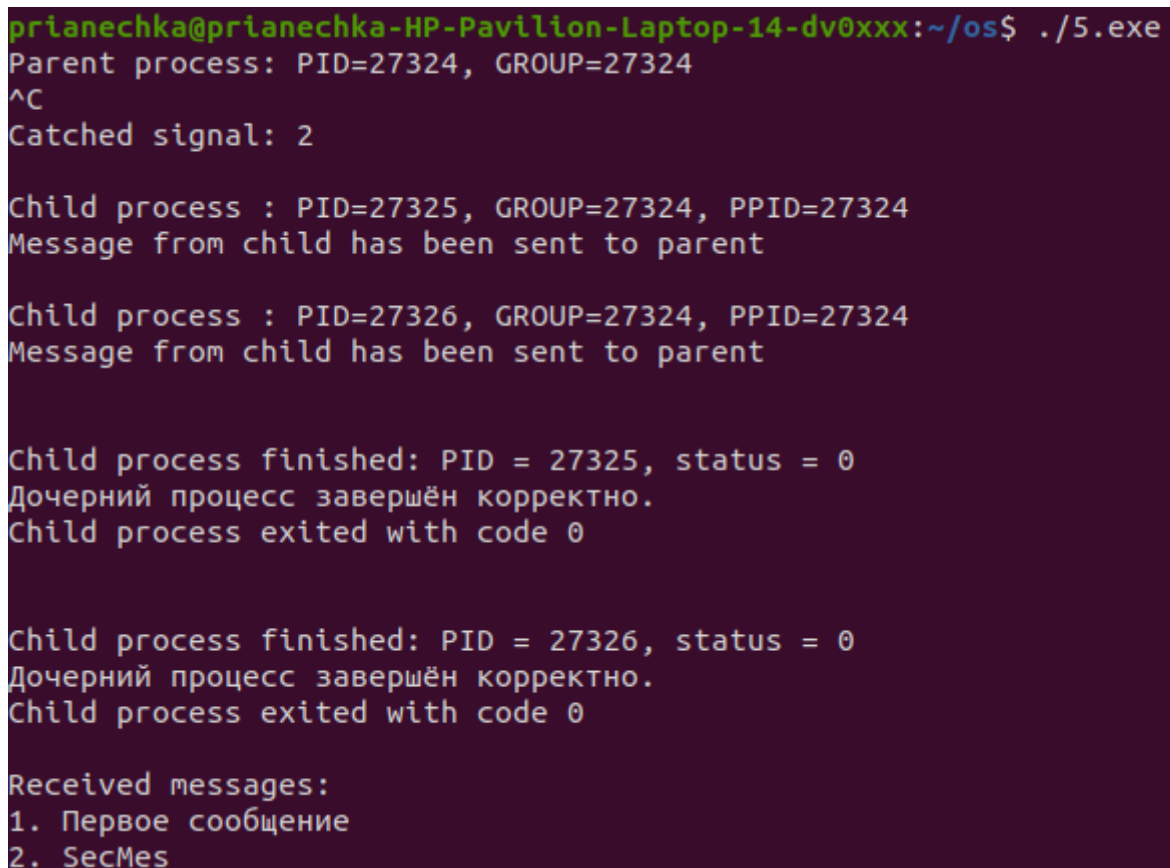
        printf("Номер сигнала: \t%d\n\n", WTERMSIG(status));
    }
    else if (WIFSTOPPED(status))
    {
        printf ("Дочерний процесс остановлен\n");
        printf ("Номер сигнала: \t%d\n\n", WSTOPSIG (status));
    }
}

close(fd[1]);
read(fd[0], buf, sizeof(buf));
printf("\nReceived messages: %s\n", buf);

return EXIT_SUCCESS;
}

```

На рисунке 6 продемонстрирован вывод программы в случае получения сигнала:



```

prianeckha@prianeckha-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./5.exe
Parent process: PID=27324, GROUP=27324
^C
Caught signal: 2

Child process : PID=27325, GROUP=27324, PPID=27324
Message from child has been sent to parent

Child process : PID=27326, GROUP=27324, PPID=27324
Message from child has been sent to parent

Child process finished: PID = 27325, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0

Child process finished: PID = 27326, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0

Received messages:
1. Первое сообщение
2. SecMes

```

Рисунок 6: Демонстрация работы программы

На рисунке 7 продемонстрирован вывод программы в случае не получения сигнала:


```
prianechka@prianechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$ ./5.exe
Parent process: PID=27344, GROUP=27344

Child process : PID=27346, GROUP=27344, PPID=27344

Child process : PID=27347, GROUP=27344, PPID=27344

Child process finished: PID = 27346, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0

Child process finished: PID = 27347, status = 0
Дочерний процесс завершён корректно.
Child process exited with code 0

Received messages:
prianechka@prianechka-HP-Pavilion-Laptop-14-dv0xxx:~/os$ █
```

Рисунок 7: Демонстрация работы программы

Конспект

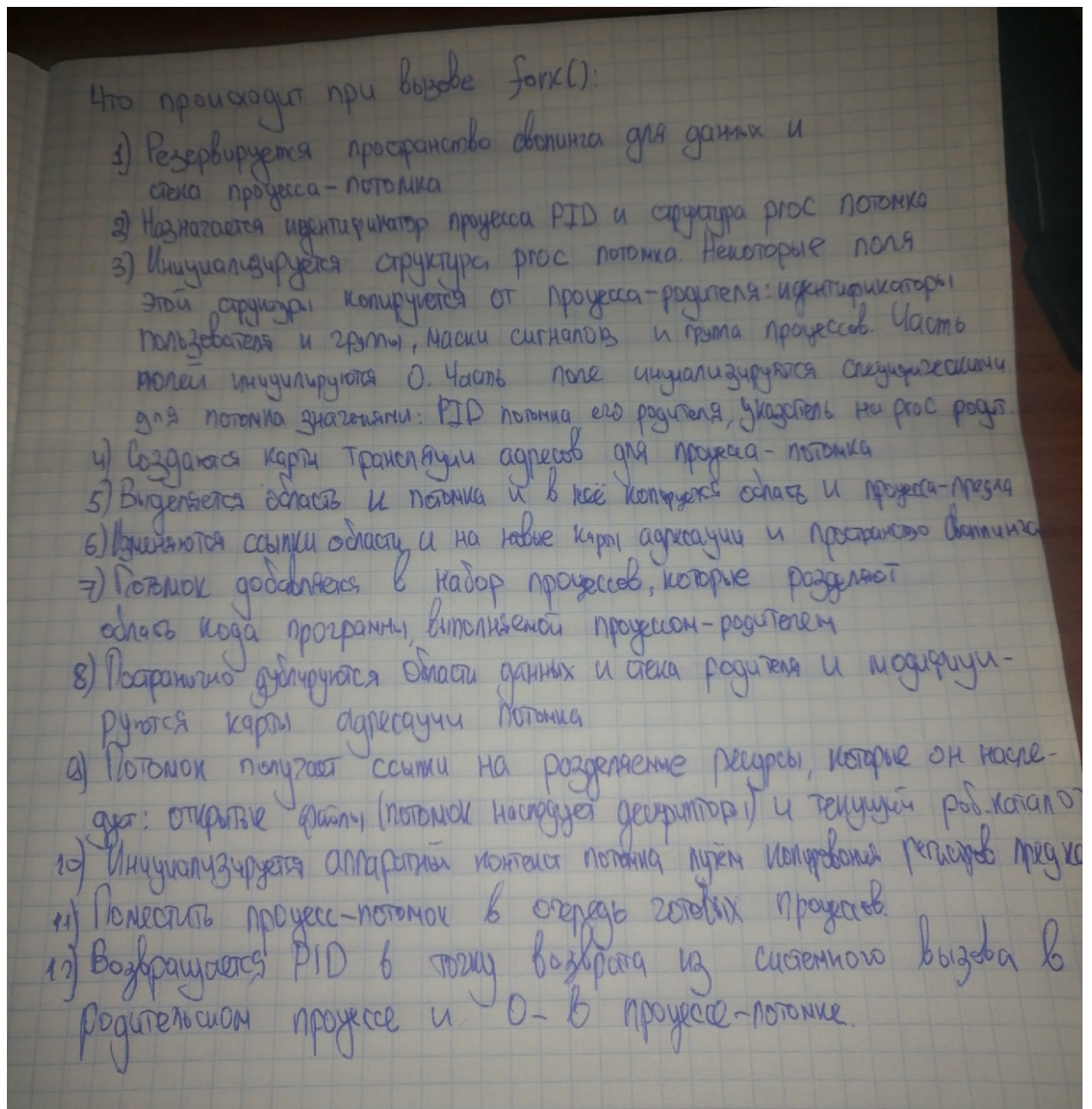


Рисунок 8: Конспект по `fork`

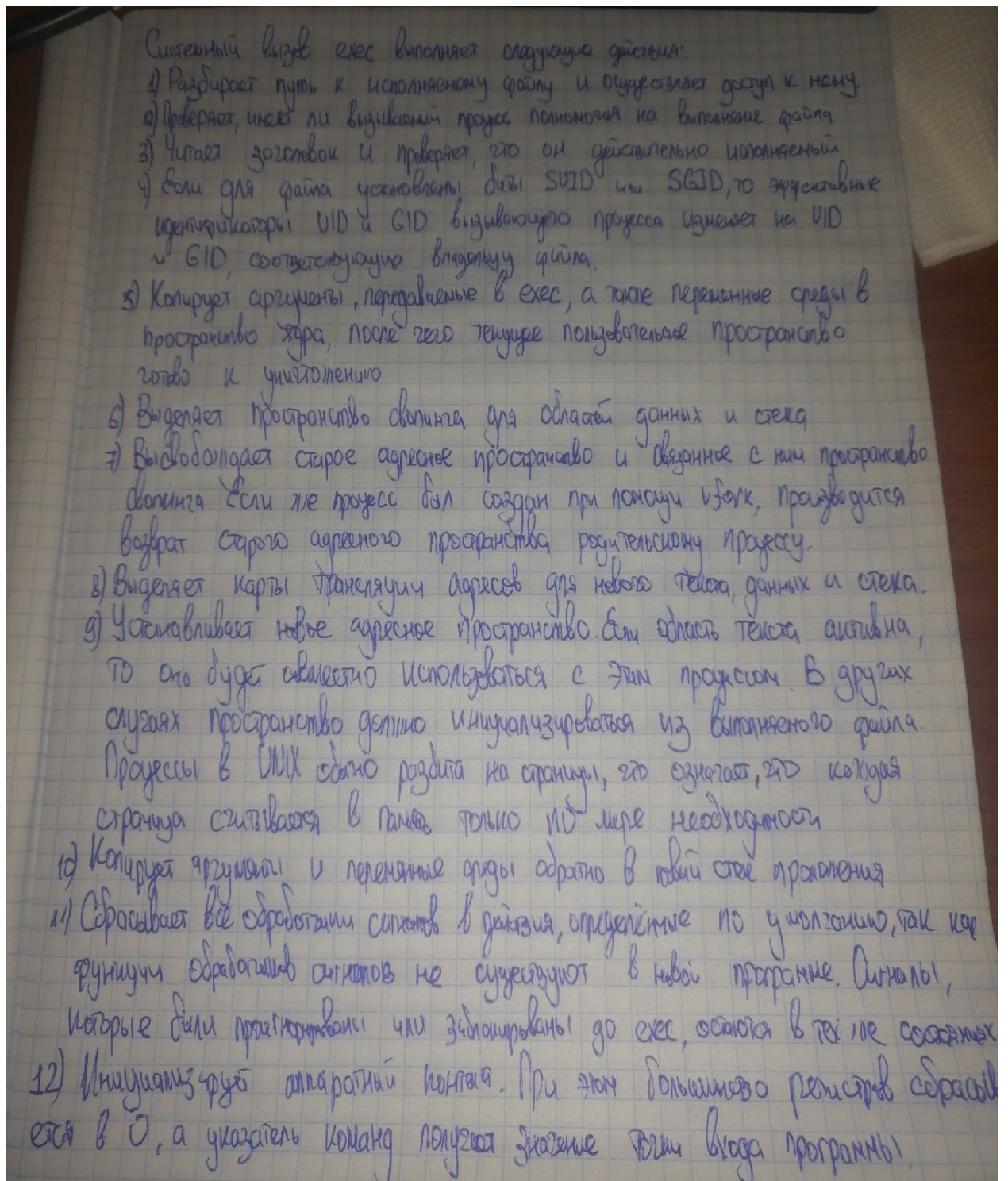


Рисунок 9: Конспект по `exec`