



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»  
на тему: «Процессы. Системные вызовы `fork()` и `exec()`»

Студент ИУ7-53Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Г. А. Миронов  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Н. Ю. Рязанова  
(И. О. Фамилия)

2021 г.

# СОДЕРЖАНИЕ

1	Задание 1	3
2	Задание 2	5
3	Задание 3	8
4	Задание 4	11
5	Задание 5	14
6	Конспект по fork и exec	18

# 1 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.

Листинг 1.1 – Листинг `main.c`. Часть 1

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

#define ERR_OK 0
#define ERR_FORK 1

#define CHILD_CNT 2
#define CHILD_SLP 2

int main()
{
    pid_t child_pids[CHILD_CNT] = {0};

    printf("parent_born: PID=%d; PPID=%d; GROUP=%d\n",
           getpid(), getppid(), getpgrp());

    for (unsigned i = 0; i < CHILD_CNT; ++i)
    {
        pid_t pid = fork();

        if (pid == -1)
        {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        }

        if (pid)
        {
            child_pids[i] = pid;
        }
        else
        {
            printf("child_%u_born: PID=%d; PPID=%d; GROUP=%d\n",
                   i, getpid(), getppid(), getpgrp());
```

## Листинг 1.2 – Листинг main.c. Часть 2

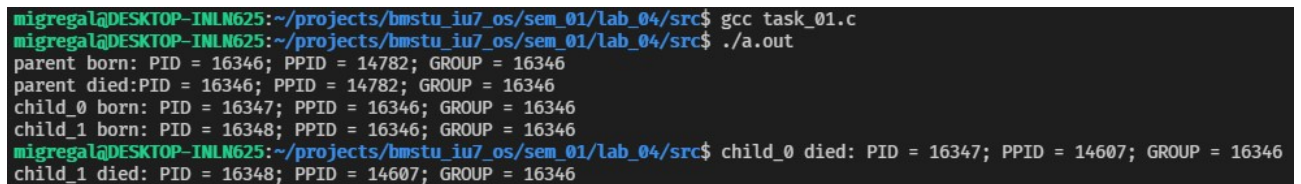
```
        sleep(CHILD_SLP);

        printf("child_%u_died: PID=%d; PPID=%d; GROUP=%d\n",
               i, getpid(), getppid(), getpgrp());

        exit(ERR_OK);
    }
}

printf("parent_died: PID=%d; PPID=%d; GROUP=%d\n",
       getpid(), getppid(), getpgrp());
}
```

На рисунке 1.1 приведен пример работы программы.



```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_01.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent born: PID = 16346; PPID = 14782; GROUP = 16346
parent died: PID = 16346; PPID = 14782; GROUP = 16346
child_0 born: PID = 16347; PPID = 16346; GROUP = 16346
child_1 born: PID = 16348; PPID = 16346; GROUP = 16346
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ child_0 died: PID = 16347; PPID = 14607; GROUP = 16346
child_1 died: PID = 16348; PPID = 14607; GROUP = 16346
```

Рисунок 1.1 – Демонстрация работы программы

## 2 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.

Листинг 2.1 – Листинг `main.c`. Часть 1

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

#define ERR_OK 0
#define ERR_FORK 1

#define CHILD_CNT 2
#define CHILD_SLP 2

int main()
{
    pid_t child_pids[CHILD_CNT] = {0};

    printf("parent_ _born:PID_=_%d;_PPID_=_%d;_PGRP_=_%d\n",
           getpid(), getppid(), getpgrp());

    for (unsigned i = 0; i < CHILD_CNT; ++i)
    {
        pid_t pid = fork();

        if (pid == -1)
        {
            fprintf(stderr, "Can't_ _fork\n");
            exit(ERR_FORK);
        }
        if (pid)
        {
            child_pids[i] = pid;
        }
        else
        {
            // child
            printf("child_%u_ _born:PID_=_%d;_PPID_=_%d;_PGRP_=_%d\n",
                   i, getpid(), getppid(), getpgrp());

            sleep(CHILD_SLP);

            printf("child_%u_ _died:PID_=_%d;_PPID_=_%d;_PGRP_=_%d\n",
```

## Листинг 2.2 – Листинг main.c. Часть 2

```
        i, getpid(), getppid(), getpgrp());

    exit(ERR_OK);
}
}

for (unsigned i = 0; i < CHILD_CNT; ++i)
{
    int status, stat_val = 0;

    printf("parent░░waiting\n");
    pid_t childpid = wait(&status);
    printf("parent░░waited:child░process░(PID░=░%d)░finished.░
        status:░%d\n",
            childpid, status);

    if (WIFEXITED(stat_val))
    {
        printf("parent░░talk:child░process░#%d░finished░with░code:░
            %d\n", i + 1,
                WEXITSTATUS(stat_val));
    }
    else if (WIFSIGNALED(stat_val))
    {
        printf("parent░░talk:child░process░#%d░finished░by░signal░
            with░code:░%d\n",
                i + 1, WTERMSIG(stat_val));
    }
    else if (WIFSTOPPED(stat_val))
    {
        printf("parent░░talk:child░process░#%d░stopped░with░code:░%
            d\n",
                i + 1, WSTOPSIG(stat_val));
    }
}

printf("parent░░died:PID░=░%d;░PPID░=░%d;░PGRP░=░%d\n",
        getpid(), getppid(), getpgrp());
}
```

На рисунке 2.1 приведен пример работы программы.

```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc ./task_02.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent  born:PID = 16574; PPID = 14782; PGRP = 16574
child_0 born:PID = 16575; PPID = 16574; PGRP = 16574
parent  waiting
child_1 born:PID = 16576; PPID = 16574; PGRP = 16574
child_1 died:PID = 16576; PPID = 16574; PGRP = 16574
child_0 died:PID = 16575; PPID = 16574; PGRP = 16574
parent  waited:child process (PID = 16575) finished. status: 0
parent  talk:child process #1 finished with code: 0
parent  waiting
parent  waited:child process (PID = 16576) finished. status: 0
parent  talk:child process #2 finished with code: 0
parent  died:PID = 16574; PPID = 14782; PGRP = 16574
```

Рисунок 2.1 – Демонстрация работы программы

### 3 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.

Листинг 3.1 – Листинг `main.c`. Часть 1

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

#define ERR_OK 0
#define ERR_FORK 1
#define ERR_EXEC 2

#define CMD_CNT 2
#define CHILD_CNT 2
#define CHILD_SLP 2
#define BUFF_SZ 2048

int main()
{
    pid_t child_pids[CHILD_CNT] = {0};
    char *cmds[CMD_CNT] = {"/lab_05", "/lab_06"};
    char *args[CMD_CNT] = {"/", NULL};

    printf("parent_born:PID=%d;PPID=%d;GROUP=%d\n",
           getpid(), getppid(), getpgrp());

    for (unsigned i = 0; i < CHILD_CNT; ++i)
    {
        pid_t pid = fork();

        if (pid == -1)
        {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        }

        if (pid)
        {
            child_pids[i] = pid;
        }
        else
    }
```



### Листинг 3.2 – Листинг main.c. Часть 2

```
{
    printf("child_%u_born:PID=%d;PPID=%d;GROUP=%d\n",
           i, getpid(), getppid(), getpgrp());

    int cmd_i = i % CMD_CNT;

    int rc = execlp(cmds[cmd_i], cmds[cmd_i],
                    args[cmd_i], (char *)NULL);

    if (rc == -1)
    {
        fprintf(stderr, "exec_failed\n");
        exit(ERR_EXEC);
    }

    assert(false);
}

for (unsigned i = 0; i < CHILD_CNT; ++i)
{
    int status, stat_val = 0;

    printf("parent_waiting\n");
    pid_t childpid = wait(&status);
    printf("parent_waited:child_process_(PID=%d)_finished;_"
           "status:%d\n",
           childpid, status);

    if (WIFEXITED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
               "finished_with_code:%d\n",
               i + 1, WEXITSTATUS(stat_val));
    }
    else if (WIFSIGNALED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
               "finished_by_signal_with_code:%d\n",
               i + 1, WTERMSIG(stat_val));
    }
    else if (WIFSTOPPED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
               "finished_stopped_with_code:%d\n",
               i + 1, WSTOPSIG(stat_val));
    }
}

printf("parent_died:PID=%d;PPID=%d;GROUP=%d\n",
       getpid(), getppid(), getpgrp());
}
```

На Рисунке 3.1 приведен пример работы программы.

```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_03.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent born:PID = 861; PPID = 9; GROUP = 861
parent waiting
child_0 born:PID = 862; PPID = 861; GROUP = 861
child_1 born:PID = 863; PPID = 861; GROUP = 861
Ant algorythm

Total files:          9
Real time spent:      16562000
Total time spent:     32
Avg filewarker time:  0
Avg digester queue time: 0.3333333333333333
Avg digester time:    3.2222222222222223
Avg queue time:       0

53986243c8b4062afc0b4847aaaa5ae6  a.out
757c0df7c6a2aed08b9ac57562489296  data/data.txt
448de48ff7023508b390f29fc3b068f5  lab_05
a533bb519a9483e38b9744b5449dfce  lab_06
5e79f755e2f32ebc8f021ef192d48eb6  task_01.c
b0f0517f1e53dc117d0ed5db1c6b28d8  task_02.c
ab81f1c9b65ccb719b4da800c3310c55  task_03.c
a86b79c6a6cfd368d8dba6b1b7c6c744  task_04.c
cc7307f1ab76d6af20fbbe39b5eee2e4  task_05.c
parent waited:child process (PID = 862) finished; status: 0
parent talk:child process #1 finished with code: 0
parent waiting
ANT ALGORITHM
+-----+
| N |      Time |
+-----+
| 2 |    8.7323ms |
| 3 |   20.5318ms |
| 4 |   35.377ms |
| 5 |   45.7493ms |
| 6 |   64.0028ms |
| 7 |  100.8108ms |
| 8 |  180.328ms |
| 9 |  198.5162ms |
|10 |  197.2043ms |
|11 |  185.7465ms |
+-----+
BRUTE ALGORITHM
+-----+
| N |      Time |
+-----+
| 2 |     5.3µs |
| 3 |     6.1µs |
| 4 |    13.7µs |
| 5 |    82.5µs |
| 6 |   352.9µs |
| 7 |    2.0128ms |
| 8 |   20.6795ms |
| 9 |   218.5122ms |
|10 |   1.5567464s |
|11 |  16.1516647s |
+-----+
parent waited:child process (PID = 863) finished; status: 0
parent talk:child process #2 finished with code: 0
parent died:PID = 861; PPID = 9; GROUP = 861
```

Рисунок 3.1 – Демонстрация работы программы

## 4 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.

Листинг 4.1 – Листинг `main.c`. Часть 1

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

#define ERR_OK 0
#define ERR_FORK 1
#define ERR_EXEC 2
#define ERR_PIPE 3

#define MSG_CNT 2
#define CHILD_CNT 2
#define CHILD_SLP 2
#define BUFF_SZ 2048

int main()
{
    int fd[2];
    char buffer[BUFF_SZ] = {0};
    pid_t child_pids[CHILD_CNT] = {0};
    char *messages[MSG_CNT] = {
        "Maybe I maybe you\n",
        "The autumn leaves of red and gold\n"};

    if (pipe(fd) == -1)
    {
        fprintf(stderr, "Can't pipe\n");
        exit(ERR_PIPE);
    }

    printf("parent born:PID=%d;PPID=%d;GROUP=%d\n",
```

## Листинг 4.2 – Листинг main.c. Часть 2

```
        getpid(), getppid(), getpgrp());

for (unsigned i = 0; i < CHILD_CNT; ++i)
{
    pid_t pid = fork();

    if (pid == -1)
    {
        fprintf(stderr, "Can't fork\n");
        exit(ERR_FORK);
    }

    if (pid)
    {
        child_pids[i] = pid;
    }
    else
    {
        printf("child_%u born:PID=%d;PPID=%d;GROUP=%d\n",
            i, getpid(), getppid(), getpgrp());

        close(fd[0]);
        write(fd[1], messages[i], strlen(messages[i]));
        printf("child_%u send:PID=%d;MSG=%s\n",
            i, getpid(), messages[i]);

        exit(ERR_OK);
    }
}

for (unsigned i = 0; i < CHILD_CNT; ++i)
{
    int status, stat_val = 0;

    printf("parent_waiting\n");
    pid_t childpid = wait(&status);
    printf("parent_waited:child_process_(PID=%d)_finished;_"
        "status:%d\n",
        childpid, status);

    if (WIFEXITED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
            "finished_with_code:%d\n",
            i + 1, WEXITSTATUS(stat_val));
    }
    else if (WIFSIGNALED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
            "finished_by_signal_with_code:%d\n",
            i + 1, WTERMSIG(stat_val));
    }
}
```

### Листинг 4.3 – Листинг main.c. Часть 3

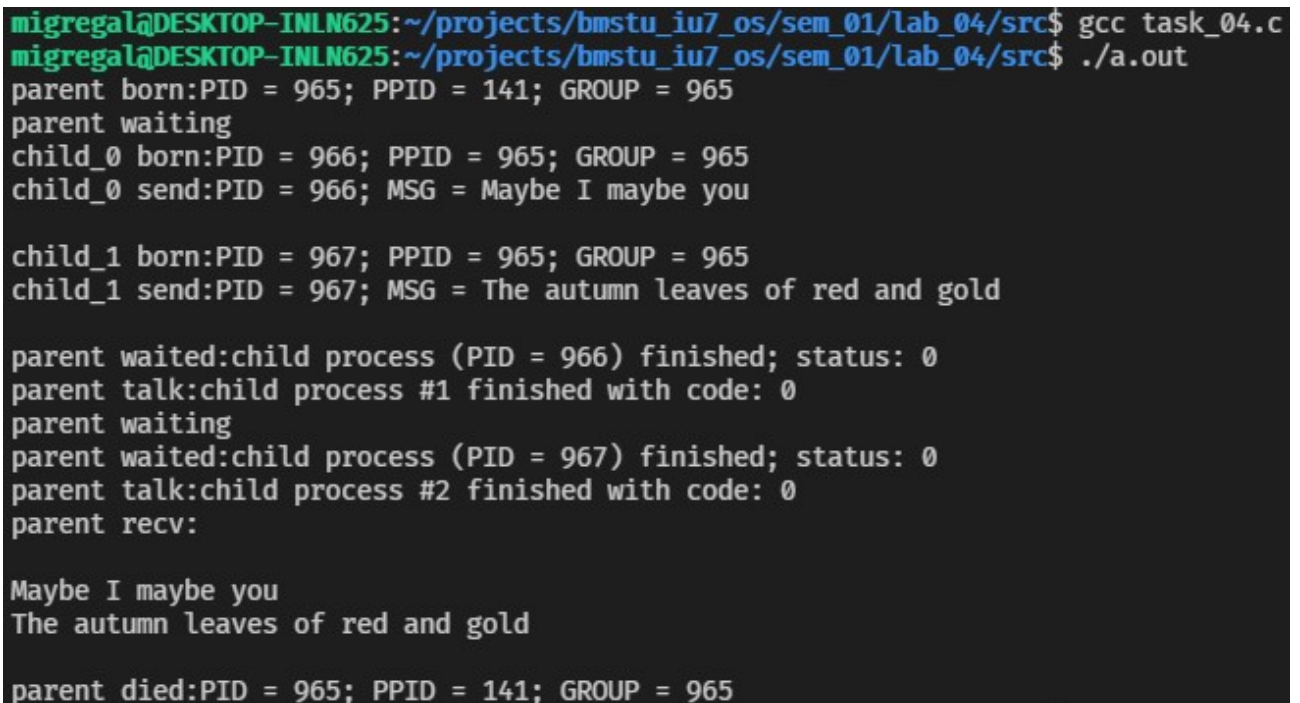
```
    }
    else if (WIFSTOPPED(stat_val))
    {
        printf("parent_talk:child_process_#%d_"
               "finished_stopped_with_code:_%d\n",
               i + 1, WSTOPSIG(stat_val));
    }
}

close(fd[1]);
ssize_t readed = read(fd[0], buffer, BUFF_SZ);

if (readed == -1)
{
    printf("error_on_read\n");
}

printf("parent_rcv:\n\n%s\n", buffer);
printf("parent_died:PID=_%d;_PPID=_%d;_GROUP=_%d\n",
        getpid(), getppid(), getpgrp());
}
```

На рисунке 4.1 приведен пример работы программы.



```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_04.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent born:PID = 965; PPID = 141; GROUP = 965
parent waiting
child_0 born:PID = 966; PPID = 965; GROUP = 965
child_0 send:PID = 966; MSG = Maybe I maybe you

child_1 born:PID = 967; PPID = 965; GROUP = 965
child_1 send:PID = 967; MSG = The autumn leaves of red and gold

parent waited:child process (PID = 966) finished; status: 0
parent talk:child process #1 finished with code: 0
parent waiting
parent waited:child process (PID = 967) finished; status: 0
parent talk:child process #2 finished with code: 0
parent rcv:

Maybe I maybe you
The autumn leaves of red and gold

parent died:PID = 965; PPID = 141; GROUP = 965
```

Рисунок 4.1 – Демонстрация работы программы

## 5 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.

Листинг 5.1 – Листинг `main.c`. Часть 1

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

#define ERR_OK 0
#define ERR_FORK 1
#define ERR_EXEC 2
#define ERR_PIPE 3

#define MSG_CNT 2
#define CHILD_CNT 2
#define CHILD_SLP 2
#define BUFF_SZ 2048

#define MOD_PRINT 0
#define MOD_QUIET 1

int mode = MOD_QUIET;

void sig_change_mod(int signum) { mode = MOD_PRINT; }

int main()
{
    signal(SIGINT, sig_change_mod);

    int fd[2];
    char buffer[BUFF_SZ] = {0};
    pid_t child_pids[CHILD_CNT] = {0};
    char *messages[MSG_CNT] = {
        "Maybe I maybe you\n",
```

## Листинг 5.2 – Листинг main.c. Часть 2

```
    "The_autumn_leaves_of_red_and_gold\n"};

if (pipe(fd) == -1)
{
    fprintf(stderr, "Can't_pipe\n");
    exit(ERR_PIPE);
}

printf("parent_born:PID=%d;PPID=%d;GROUP=%d\n",
        getpid(), getppid(), getpgrp());

for (unsigned i = 0; i < CHILD_CNT; ++i, sleep(2))
{
    pid_t pid = fork();

    if (pid == -1)
    {
        fprintf(stderr, "Can't_fork\n");
        exit(ERR_FORK);
    }
    if (pid)
    {
        child_pids[i] = pid;
    }
    else
    {
        printf("child_%u_born:PID=%d;PPID=%d;GROUP=%d\n",
                i, getpid(), getppid(), getpgrp());

        if (mode == MOD_PRINT)
        {
            close(fd[0]);
            write(fd[1], messages[i], strlen(messages[i]));
            printf("child_%u_send:PID=%d;MSG=%s\n",
                    i, getpid(), messages[i]);
        }
        else
        {
            printf("child_%u_send:quiet_mode_not_message_send\n", i);
        }

        exit(ERR_OK);
    }
}

for (unsigned i = 0; i < CHILD_CNT; ++i)
{
    int status, stat_val = 0;

    printf("parent_waiting\n");
```

### Листинг 5.3 – Листинг main.c. Часть 3

```
pid_t childpid = wait(&status);
printf("parent_waited:child_process_(PID=%d)_finished;_"
       "status:_%d\n",
       childpid, status);

if (WIFEXITED(stat_val))
{
    printf("parent_talk:child_process_#%d_"
           "finished_with_code:_%d\n",
           i + 1, WEXITSTATUS(stat_val));
}
else if (WIFSIGNALED(stat_val))
{
    printf("parent_talk:child_process_#%d_"
           "finished_by_signal_with_code:_%d\n",
           i + 1, WTERMSIG(stat_val));
}
else if (WIFSTOPPED(stat_val))
{
    printf("parent_talk:child_process_#%d_"
           "finished_stopped_with_code:_%d\n",
           i + 1, WSTOPSIG(stat_val));
}
}

close(fd[1]);
ssize_t readed = read(fd[0], buffer, BUFF_SZ);

if (readed == -1)
    printf("error_on_read\n");

printf("parent_recv:\n\n%s\n", buffer);
printf("parent_died:PID=%d;_PPID=%d;_GROUP=%d\n",
       getpid(), getppid(), getpgrp());
}
```



На рисунке 5.1 приведен пример работы программы без вызова сигнала.

```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_05.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent born:PID = 1166; PPID = 141; GROUP = 1166
child_0 born:PID = 1167; PPID = 1166; GROUP = 1166
child_0 send:quiet mode not message send
child_1 born:PID = 1179; PPID = 1166; GROUP = 1166
child_1 send:quiet mode not message send
parent waiting
parent waited:child process (PID = 1167) finished; status: 0
parent talk:child process #1 finished with code: 0
parent waiting
parent waited:child process (PID = 1179) finished; status: 0
parent talk:child process #2 finished with code: 0
parent recv:

parent died:PID = 1166; PPID = 141; GROUP = 1166
```

Рисунок 5.1 – Демонстрация работы программы (сигнал не вызывается)

На рисунке 5.2 приведен пример работы программы с вызовом сигнала.

```
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc ./task_05.c
migregal@DESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ ./a.out
parent born:PID = 656; PPID = 185; GROUP = 656
child_0 born:PID = 657; PPID = 656; GROUP = 656
child_0 send:quiet mode not message send
^Cchild_1 born:PID = 658; PPID = 656; GROUP = 656
child_1 send:PID = 658; MSG = The autumn leaves of red and gold

parent waiting
parent waited:child process (PID = 657) finished; status: 0
parent talk:child process #1 finished with code: 0
parent waiting
parent waited:child process (PID = 658) finished; status: 0
parent talk:child process #2 finished with code: 0
parent recv:

The autumn leaves of red and gold

parent died:PID = 656; PPID = 185; GROUP = 656
```

Рисунок 5.2 – Демонстрация работы программы (сигнал вызывается)

## 6 Конспект по fork и exec

На рис. 6.1 приведен конспект по системному вызову `fork()`.

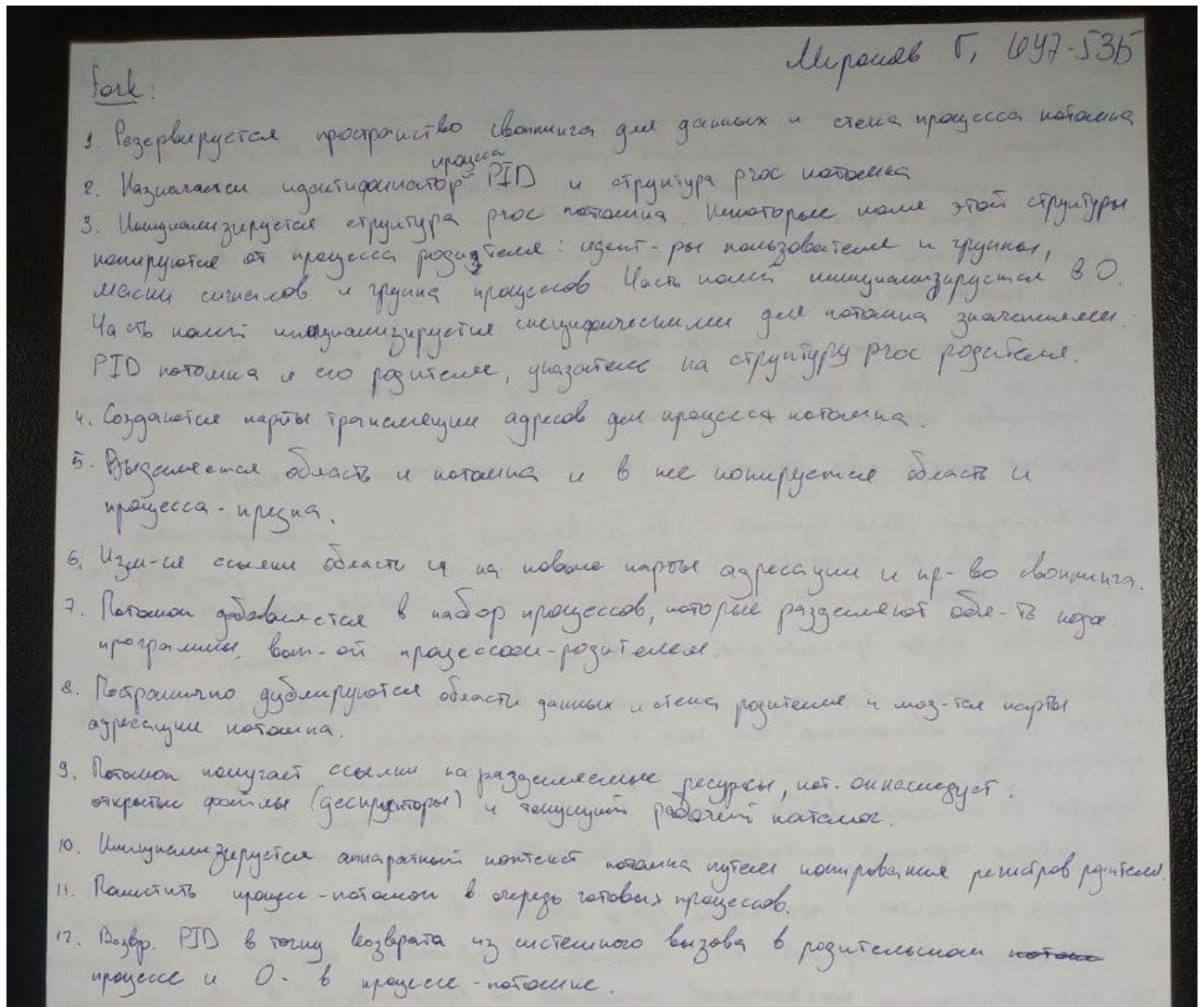


Рисунок 6.1 – Конспект по системному вызову `fork()`

На рис. 6.2 приведен конспект по системному вызову `exec()`.

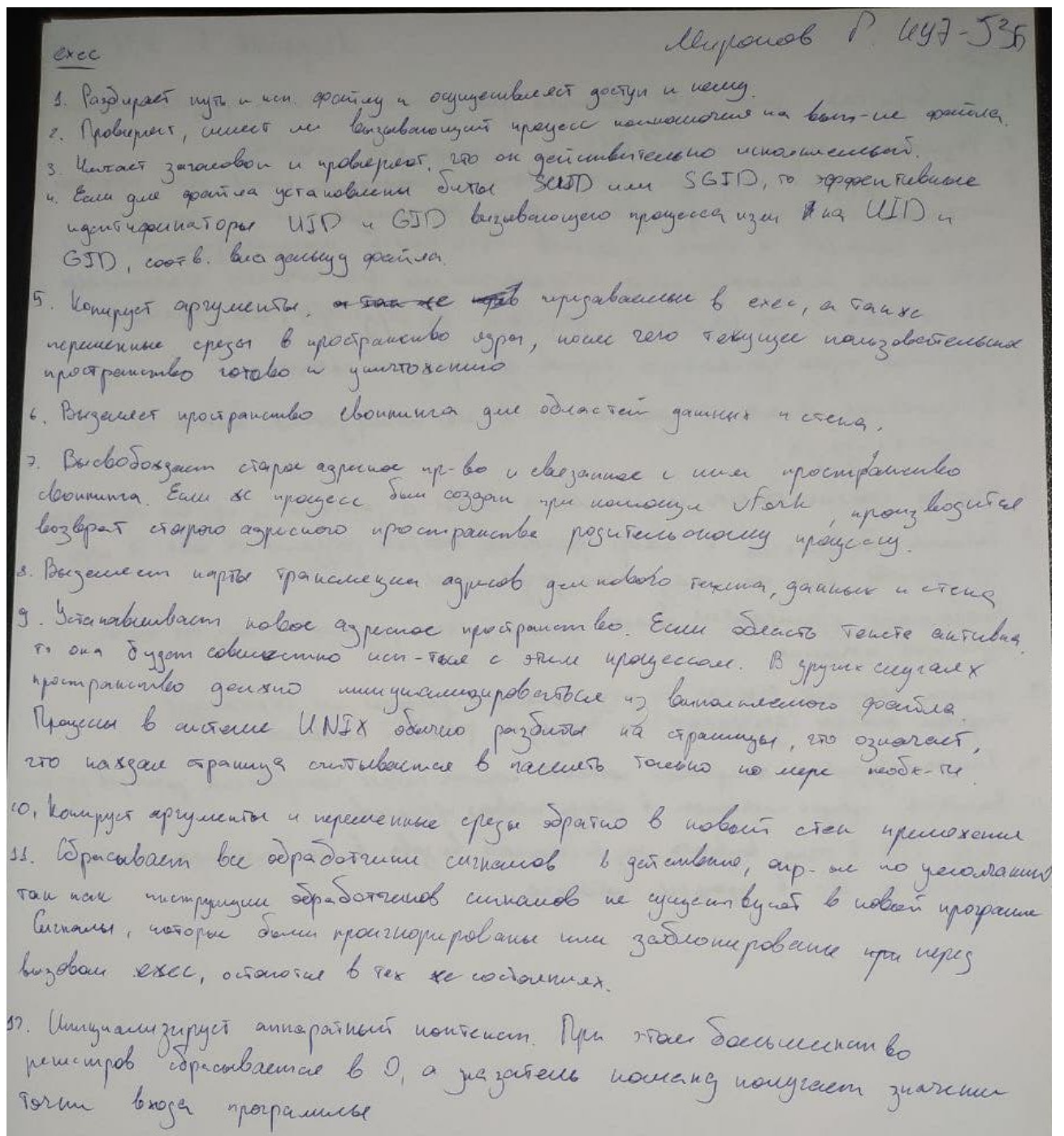


Рисунок 6.2 – Конспект по системному вызову `exec()`