



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)

Д. В. Недолужко
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Н. Ю. Рязанова
(И. О. Фамилия)

2021 г.

1 Теоретический раздел

ИУ7-53Б Кедошукис Д.

Выполнение вызова fork()

1. Резервируется пространство памяти для данных и стека процесса-потомка
2. Инициализируется идентификатор процесса PID и структура ядра процесса
3. Инициализируется структура ядра процесса. Некоторые поля этой структуры копируются от процесса-родителя: идентификатор пользователя и группы, имена сигналов и группа процессов. Часть полей инициализируется 0. Часть полей инициализируется специфическими для процесса значениями: PID процесса и его родителя, указатель на структуру ядра родителя
4. Создается карта трансляции адресов для процесса-потомка
5. Выделяется область и процесс и в нее копируется и передается процесс
6. Инициализируются ссылки области и на новые карты адресации и ядро области
7. Процесс добавляется в набор процессов, которые разделяют область кода процесса, выполняемой процессом-родителем
8. Постранично формируются области данных и стека родителя и идентифицируются карты адресации процесса
9. Процесс получает ссылки на разделяемые ресурсы, которые он наследует: открытые файлы (потомок наследует дескрипторы) и таблицы таблиц имен.
10. Инициализируется аппаратный контекст процесса путем копирования регистров родителя
11. Помещается процесс-потомок в очередь готовых процессов
12. Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 в процессе-потомке.

Рисунок 1.1 – Действия системного вызова fork

Системный вызов `exec` выполняет следующие действия

1. Разбирает путь к исполняемому файлу и организует загрузку ядра
2. Проверяет, имеет ли вызывающий процесс полномочия на выполнение файла.
3. Читает заголовки и проверяет, что он действительно исполняемый
4. Если для файла установлен биты `SUID` или `SGID`, то соответствующие идентификаторы `UID` и `GID` вызывающего процесса заменяют на `UID` и `GID`, соответствующие владельцу файла
5. Копирует аргументы, переданные в `exec`, а также те переменные среды в пространство ядра, после чего запускает выполняемое пространство ядра и уничтожение.
6. Выделяет пространство ядра для данных, данных и стека
7. Выделяет старое адресное пространство и связывает с ним пространство ядра. Если же процесс был создан при помощи `vfork`, производится возврат старого адресного пространства родительскому процессу.
8. Выделяет карты привязки адресов для нового текста, данных и стека
9. Устанавливает новое ядро ядра. Если данные текста инициализированы (то ядром процесс выполняет ту же программу), то она будет автоматически исполняться с теми же процессами. В других случаях пространство должно инициализироваться из испол. файла. Процесс в системе `UNIX` обычно разбит на страницы, что означает, что каждая страница снимается в память только по мере необходимости
10. Копируются аргументы и переменные среды обратно к ядру и новый стек приводект.
11. Собираются сигналы в действии, определенные по умолчанию, т.е. функции обработки не вызваны в новой ядре. Сигналы, которые приток или заблокированы перед вызовом `exec` остаются в ядре не сошедших.
12. Инициализирует аппаратный контекст. При этом данные регистров сохраняются в 0, а указатель команд получает значение точки для входа программы.

Рисунок 1.2 – Действия системного вызова `exec`

2 Практический раздел

2.1 Задание 1

Листинг 2.1 – Листинг программы 1. Часть 1

```
#include <stdio.h>    // printf, fprintf, sleep
#include <stdlib.h>    // exit
#include <unistd.h>    // fork

#define ERR_OK 0
#define ERR_FORK 1

#define CHILD_CNT 2
#define CHILD_SLP 2

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

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

    for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_i) {
        int pid = fork();

        if (pid == -1) {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        } else if (pid == 0) {
            // child
            printf("child%u_born_: PID=%d; PPID=%d; GROUP=%d\n"
                ↪ , child_i,
                getpid(), getppid(), getpgrp());

            sleep(CHILD_SLP);

            printf("child%u_died_: PID=%d; PPID=%d; GROUP=%d\n"
                ↪ , child_i,
                getpid(), getppid(), getpgrp());

            exit(ERR_OK);
        } else {
            // parent
            child_pids[child_i] = pid;
            printf("parent_mess_: PID=%d; CHILD_PID=%d\n", getpid
                ↪ (), pid);
        }
    }
}
```

Листинг 2.2 – Листинг программы 1. Часть 2

```
printf("parent_died: PID=%d; PPID=%d; GROUP=%d\n",  
    ↪ getpid(),  
        getppid(), getpgrp());  
  
return 0;  
}
```

```
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./01  
parent born : PID = 14629 ; PPID = 11686 ; GROUP = 14629  
parent mess : PID = 14629 ; CHILD_PID = 14630  
parent mess : PID = 14629 ; CHILD_PID = 14631  
child0 born : PID = 14630 ; PPID = 14629 ; GROUP = 14629  
parent died : PID = 14629 ; PPID = 11686 ; GROUP = 14629  
child1 born : PID = 14631 ; PPID = 14629 ; GROUP = 14629  
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ child0 died : PID = 14630 ; PPID = 1 ; GROUP = 14629  
child1 died : PID = 14631 ; PPID = 1 ; GROUP = 14629
```

Рисунок 2.1 – Пример работы программы 1

Программа запускает два новых процесса системным вызовом `fork()`. Процессы выводят собственные идентификаторы, идентификаторы предков, идентификаторы групп. Процесс предок также выводит идентификаторы потомков отдельным сообщением. После смерти предка потомки выподят свои сообщения повторно, идентификатор предка при этом равен единице.

2.2 Задание 2

Листинг 2.3 – Листинг программы 2. Часть 1

```
#include <stdio.h>      // printf, fprintf, sleep
#include <stdlib.h>     // exit
#include <sys/wait.h>    // wait
#include <unistd.h>     // fork

#define ERR_OK 0
#define ERR_FORK 1

#define CHILD_CNT 2
#define CHILD_SLP 2

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

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

    for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_i) {
        int pid = fork();

        if (pid == -1) {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        } else if (pid == 0) {
            // child
            printf("child%u_ born_: PID=%d; PPID=%d; GROUP=%d\n"
                ↪ , child_i,
                   getpid(), getppid(), getpgrp());

            sleep(CHILD_SLP);

            printf("child%u_ died_: PID=%d; PPID=%d; GROUP=%d\n"
                ↪ , child_i,
                   getpid(), getppid(), getpgrp());

            exit(ERR_OK);
        } else {
            // parent
            child_pids[child_i] = pid;
        }
    }

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

Листинг 2.4 – Листинг программы 2. Часть 2

```
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",
        child_i + 1, WEXITSTATUS(stat_val));
} else if (WIFSIGNALED(stat_val)) {
    printf(
        "parent_talk: child_process #%d finished by signal
    ↳ with code: %d\n",
        child_i + 1, WTERMSIG(stat_val));
} else if (WIFSTOPPED(stat_val)) {
    printf("parent_talk: child_process #%d finished stopped
    ↳ with code: %d\n",
        child_i + 1, WSTOPSIG(stat_val));
}
}

return 0;
}
```

```
parent born : PID = 5314 ; PPID = 3640 ; GROUP = 5314
parent waiting
child0 born : PID = 5315 ; PPID = 5314 ; GROUP = 5314
child1 born : PID = 5316 ; PPID = 5314 ; GROUP = 5314
child0 died : PID = 5315 ; PPID = 5314 ; GROUP = 5314
child1 died : PID = 5316 ; PPID = 5314 ; GROUP = 5314
parent waited : child process (PID = 5315) finished. status: 0
parent talk : child process #1 finished with code: 0
parent waiting
parent waited : child process (PID = 5316) finished. status: 0
parent talk : child process #2 finished with code: 0
```

Рисунок 2.2 – Пример работы программы 2

Программа работает по схеме первого задания. Но перед смертью процесс предок дожидается завершения процессов-потомков и выводит статус их завершения.

2.3 Задание 3

Листинг 2.5 – Листинг программы 3. Часть 1

```
#include <stdbool.h>    // false
#include <stdio.h>      // printf, fprintf, sleep
#include <stdlib.h>     // exit, NULL
#include <sys/wait.h>   // wait
#include <unistd.h>     // fork, execvp

#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() {
    int child_pids[CHILD_CNT] = {0};
    char* cmds[CMD_CNT] = {"/home/deniska/dev/bmstu_iu7_oop/lab_03/
        ↪ build/lab_03",
                           "/home/deniska/dev/bmstu_iu7_cgcp/build/
        ↪ raytracing"};

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

    for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_i) {
        int pid = fork();

        if (pid == -1) {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        } else if (pid == 0) {
            // child
            printf("child%u_ born_: PID= %d; PPID= %d; GROUP= %d\n"
                ↪ , child_i,
                getpid(), getppid(), getpgrp());

            int cmd_i = child_i % CMD_CNT;

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

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


Листинг 2.6 – Листинг программы 3. Часть 2

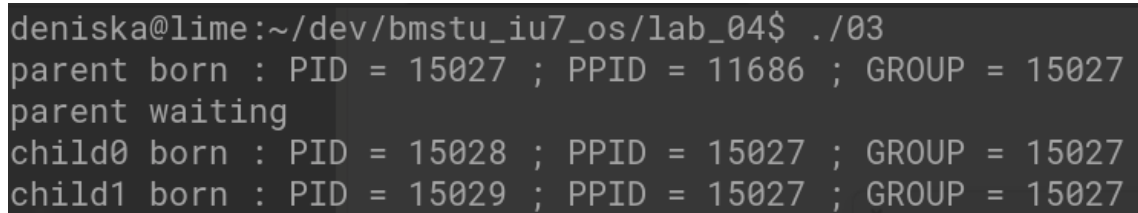
```
    } else {
        // parent
        child_pids[child_i] = pid;
    }
}

for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_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",
            child_i + 1, WEXITSTATUS(stat_val));
    } else if (WIFSIGNALED(stat_val)) {
        printf(
            "parent_talk: child_process #d finished by signal
            ↪ with code: %d\n",
            child_i + 1, WTERMSIG(stat_val));
    } else if (WIFSTOPPED(stat_val)) {
        printf("parent_talk: child_process #d finished stopped
        ↪ with code: %d\n",
            child_i + 1, WSTOPSIG(stat_val));
    }
}

return 0;
}
```



```
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./03
parent born : PID = 15027 ; PPID = 11686 ; GROUP = 15027
parent waiting
child0 born : PID = 15028 ; PPID = 15027 ; GROUP = 15027
child1 born : PID = 15029 ; PPID = 15027 ; GROUP = 15027
```

Рисунок 2.3 – Пример работы программы 3. Процесс запустил потомков

Создаются 2 процесса потомка. Процесс-потомок вызывает системный вызов `exes()`, после чего запускается приложение с графическим интерфейсом, а процесс-предок ждет завершения процесса-потомка.

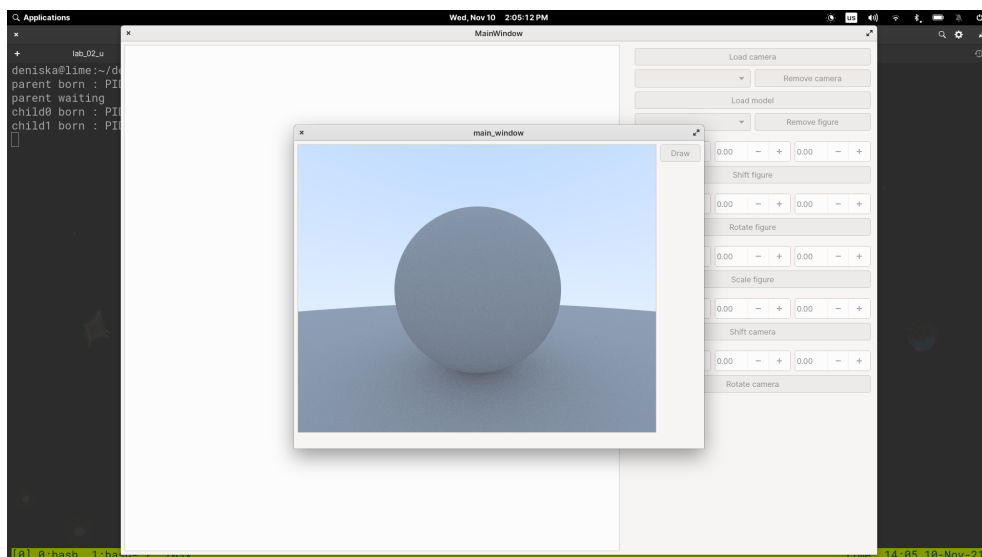


Рисунок 2.4 – Пример работы программы 3. Запущенное приложение потомок со сферами

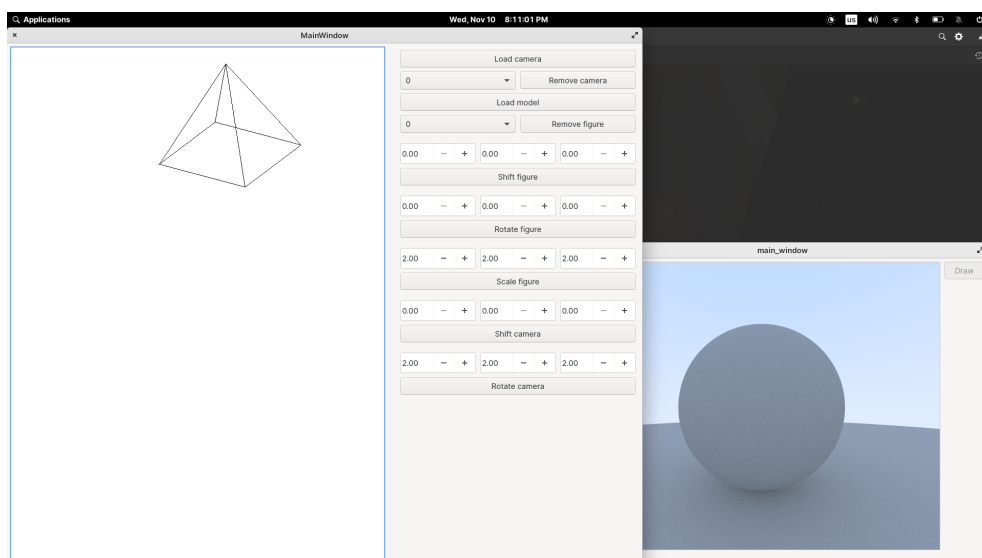


Рисунок 2.5 – Пример работы программы 3. Запущенное приложение потомок с каркасной моделью

```
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./03
parent born : PID = 15027 ; PPID = 11686 ; GROUP = 15027
parent waiting
child0 born : PID = 15028 ; PPID = 15027 ; GROUP = 15027
child1 born : PID = 15029 ; PPID = 15027 ; GROUP = 15027
parent waited : child process (PID = 15029) finished. status: 0
parent talk : child process #1 finished with code: 0
parent waiting
parent waited : child process (PID = 15028) finished. status: 0
parent talk : child process #2 finished with code: 0
```

Рисунок 2.6 – Пример работы программы 3. Вывод программы после завершения потомков

2.4 Задание 4

Листинг 2.7 – Листинг программы 4. Часть 1

```
#include <stdbool.h>    // false
#include <stdio.h>      // printf, fprintf, sleep
#include <stdlib.h>     // exit, NULL
#include <string.h>     // strlen
#include <sys/wait.h>   // wait
#include <unistd.h>     // fork, execlp, read, write, ssize_t,
    ↪ pipe

#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};
    int child_pids[CHILD_CNT] = {0};
    char* messages[MSG_CNT] = {
        "У окна дождь расскажет мне тайком, "
        "Как он жил вчера, как он будет жить потом. "
        "Я найду путь, который искал, ",
        "Дождь возьмет меня туда, где луна висит меж скал. "};

    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 child_i = 0; child_i < CHILD_CNT; ++child_i) {
        int pid = fork();

        if (pid == -1) {
            fprintf(stderr, "Can't fork\n");
            exit(ERR_FORK);
        } else if (pid == 0) {
            // child
```

Листинг 2.8 – Листинг программы 4. Часть 2

```

    printf("child%u_born:_PID=_%d;_PPID=_%d;_GROUP=_%d\n"
        ↪ , child_i,
            getpid(), getppid(), getpgrp());

    int msg_i = child_i % MSG_CNT;
    close(fd[0]);
    write(fd[1], messages[msg_i], strlen(messages[msg_i]));
    printf("child%u_send:_PID=_%d;_MSG=_%s\n", child_i,
        ↪ getpid(),
            messages[msg_i]);

    exit(ERR_OK);
} else {
    // parent
    child_pids[child_i] = pid;
}
}

for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_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",
            child_i + 1, WEXITSTATUS(stat_val));
    } else if (WIFSIGNALED(stat_val)) {
        printf(
            "parent_talk:_child_process_#%d_finished_by_signal_
            ↪ with_code:_%d\n",
            child_i + 1, WTERMSIG(stat_val));
    } else if (WIFSTOPPED(stat_val)) {
        printf("parent_talk:_child_process_#%d_finished_stopped_
            ↪ with_code:_%d\n",
            child_i + 1, WSTOPSIG(stat_val));
    }
}

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

if (readed == -1) {

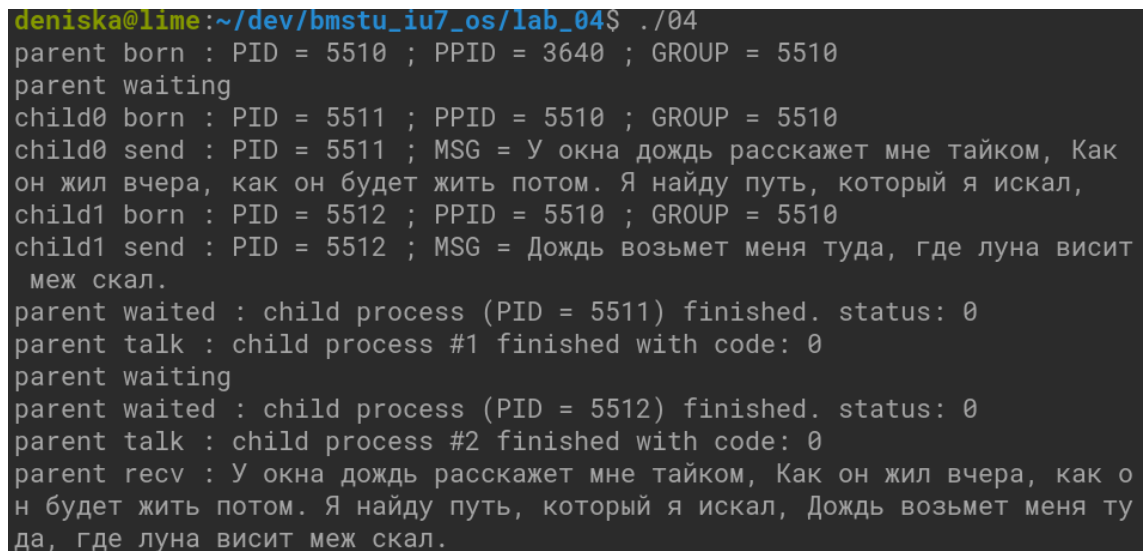
```

Листинг 2.9 – Листинг программы 4. Часть 2

```
    printf("error on read\n");
}

printf("parent recv: %s\n", buffer);

return 0;
}
```



```
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./04
parent born : PID = 5510 ; PPID = 3640 ; GROUP = 5510
parent waiting
child0 born : PID = 5511 ; PPID = 5510 ; GROUP = 5510
child0 send : PID = 5511 ; MSG = У окна дождь расскажет мне тайком, Как
он жил вчера, как он будет жить потом. Я найду путь, который я искал,
child1 born : PID = 5512 ; PPID = 5510 ; GROUP = 5510
child1 send : PID = 5512 ; MSG = Дождь возьмет меня туда, где луна висит
меж скал.
parent waited : child process (PID = 5511) finished. status: 0
parent talk : child process #1 finished with code: 0
parent waiting
parent waited : child process (PID = 5512) finished. status: 0
parent talk : child process #2 finished with code: 0
parent recv : У окна дождь расскажет мне тайком, Как он жил вчера, как о
н будет жить потом. Я найду путь, который я искал, Дождь возьмет меня ту
да, где луна висит меж скал.
```

Рисунок 2.7 – Пример работы программы 4

В программе запускаются 2 процесса потомка. Каждый из них пишет в неименованный канал уникальное сообщения. Процесс предок читает из канала сообщения и выводит их на экран.

2.5 Задание 5

Листинг 2.10 – Листинг программы 5. Часть 1

```
#include <stdbool.h>    // false
#include <stdio.h>      // printf, fprintf, sleep
#include <stdlib.h>     // exit, NULL
#include <string.h>     // strlen
#include <sys/wait.h>   // wait
#include <unistd.h>     // fork, execlp, read, write, ssize_t,
    ↪ pipe

#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_PRINT;

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

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

    int fd[2];
    char buffer[BUFF_SZ] = {0};
    int child_pids[CHILD_CNT] = {0};
    char* messages[MSG_CNT] = {
        "У окна дождь расскажет мне тайком, "
        "Как он жил вчера, как он будет жить потом. "
        "Я найду путь, который искал, ",
        "Дождь возьмет меня туда, где луна висит меж скал. "};

    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 child_i = 0; child_i < CHILD_CNT; ++child_i,
    ↪ sleep(2)) {
    int pid = fork();

    if (pid == -1) {
        fprintf(stderr, "Can't fork\n");
        exit(ERR_FORK);
    } else if (pid == 0) {
        // child
        printf("child%u born: PID=%d; PPID=%d; GROUP=%d\n"
            ↪ , child_i,
                getpid(), getppid(), getpgrp());

        int msg_i = child_i % MSG_CNT;
        if (mode == MOD_PRINT) {
            close(fd[0]);
            write(fd[1], messages[msg_i], strlen(messages[msg_i]));
            printf("child%u send: PID=%d; MSG=%s\n", child_i,
                ↪ getpid(),
                    messages[msg_i]);
        } else {
            printf("child%u send: quiet mode: not message send\n",
                ↪ child_i);
        }

        exit(ERR_OK);
    } else {
        // parent
        child_pids[child_i] = pid;
    }
}

for (unsigned child_i = 0; child_i < CHILD_CNT; ++child_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",
                child_i + 1, WEXITSTATUS(stat_val));
    } else if (WIFSIGNALED(stat_val)) {

```

Листинг 2.12 – Листинг программы 5. Часть 3

```

        printf(
            "parent_talk:_child_process_#%d_finished_by_signal_
            ↪ with_code:_%d\n",
            child_i + 1, WTERMSIG(stat_val));
    } else if (WIFSTOPPED(stat_val)) {
        printf("parent_talk:_child_process_#%d_finished_stopped_
        ↪ with_code:_%d\n",
            child_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:_%s\n", buffer);

return 0;
}

```

```

deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./05
parent born : PID = 5597 ; PPID = 3640 ; GROUP = 5597
child0 born : PID = 5598 ; PPID = 5597 ; GROUP = 5597
child0 send : PID = 5598 ; MSG = У окна дождь расскажет мне тайком, Как
он жил вчера, как он будет жить потом. Я найду путь, который я искал,
child1 born : PID = 5599 ; PPID = 5597 ; GROUP = 5597
child1 send : PID = 5599 ; MSG = Дождь возьмет меня туда, где луна висит
меж скал.
parent waiting
parent waited : child process (PID = 5598) finished. status: 0
parent talk : child process #1 finished with code: 0
parent waiting
parent waited : child process (PID = 5599) finished. status: 0
parent talk : child process #2 finished with code: 0
parent recv : У окна дождь расскажет мне тайком, Как он жил вчера, как о
н будет жить потом. Я найду путь, который я искал, Дождь возьмет меня ту
да, где луна висит меж скал.

```

Рисунок 2.8 – Пример работы программы 5 без отправки сигнала

В программе запускаются 2 процесса потомка. Каждый из них пишет в неименованный канал уникальные сообщения. Процесс предок читает из канала сообщения и выводит их на экран. Если сигнал *SIGINT* послан, то устанавливается тихий режим, при котором процессы-потомки не передают сообщения через канал.

```
deniska@lime:~/dev/bmstu_iu7_os/lab_04$ ./05
parent born : PID = 5619 ; PPID = 3640 ; GROUP = 5619
child0 born : PID = 5620 ; PPID = 5619 ; GROUP = 5619
child0 send : PID = 5620 ; MSG = У окна дождь расскажет мне тайком, Как
он жил вчера, как он будет жить потом. Я найду путь, который я искал,
^Cchild1 born : PID = 5621 ; PPID = 5619 ; GROUP = 5619
child1 send : quiet mode : not message send
parent waiting
parent waited : child process (PID = 5620) finished. status: 0
parent talk : child process #1 finished with code: 0
parent waiting
parent waited : child process (PID = 5621) finished. status: 0
parent talk : child process #2 finished with code: 0
parent recv : У окна дождь расскажет мне тайком, Как он жил вчера, как о
н будет жить потом. Я найду путь, который я искал,
```

Рисунок 2.9 – Пример работы программы 5 с посылкой сигнала