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 No4 по дисциплине "Операционные системы"

Тема Процессы. Системные вызовы fork() и exec()

Студент Каримзай А-Х.

Группа ИУ7и-55Б

Оценка (баллы) ______________________________

Преподаватели Рязанова Н.Ю.

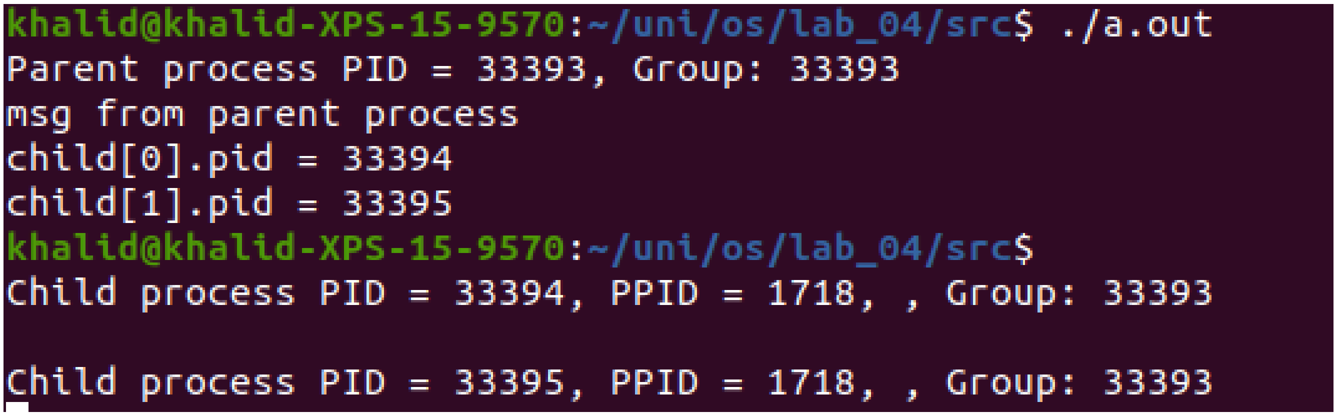
Москва — 2021 г.

Задание No1

Процессы-сироты. В программе создаются не менее двух потомков. В потомках вызывается sleep(). Чтобы предок гарантированно завершился раньше своих помков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе.

Листинг 1: Процессы-сироты

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

#define REP_N 2 // loop rep time
#define SLP_T 2 // sleep time

enum error_code
{
    ok,
    error
};

int main()
{
    int child[REP_N];

    printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());

    for (size_t i = 0; i < REP_N; i++) {
        pid_t pid = fork();

        if (pid == -1) {
            perror("Can't fork!");
            return error;
        } else if (pid == 0) {
            sleep(SLP_T);
            printf("\nChild process PID = %d, PPID = %d, , Group: %d\n", \
                                    getpid(), getppid(), getpgrp());
            return ok;
        } else {
            child[i] = pid;
        }

    }
    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++)
        printf("child[%zu].pid = %d\n", i, child[i]);

    return ok;
}
```



```
khalid@khalid-XPS-15-9570:~/uni/os/lab_04/src$ ./a.out
Parent process PID = 33393, Group: 33393
msg from parent process
child[0].pid = 33394
child[1].pid = 33395
khalid@khalid-XPS-15-9570:~/uni/os/lab_04/src$
Child process PID = 33394, PPID = 1718, , Group: 33393

Child process PID = 33395, PPID = 1718, , Group: 33393
```

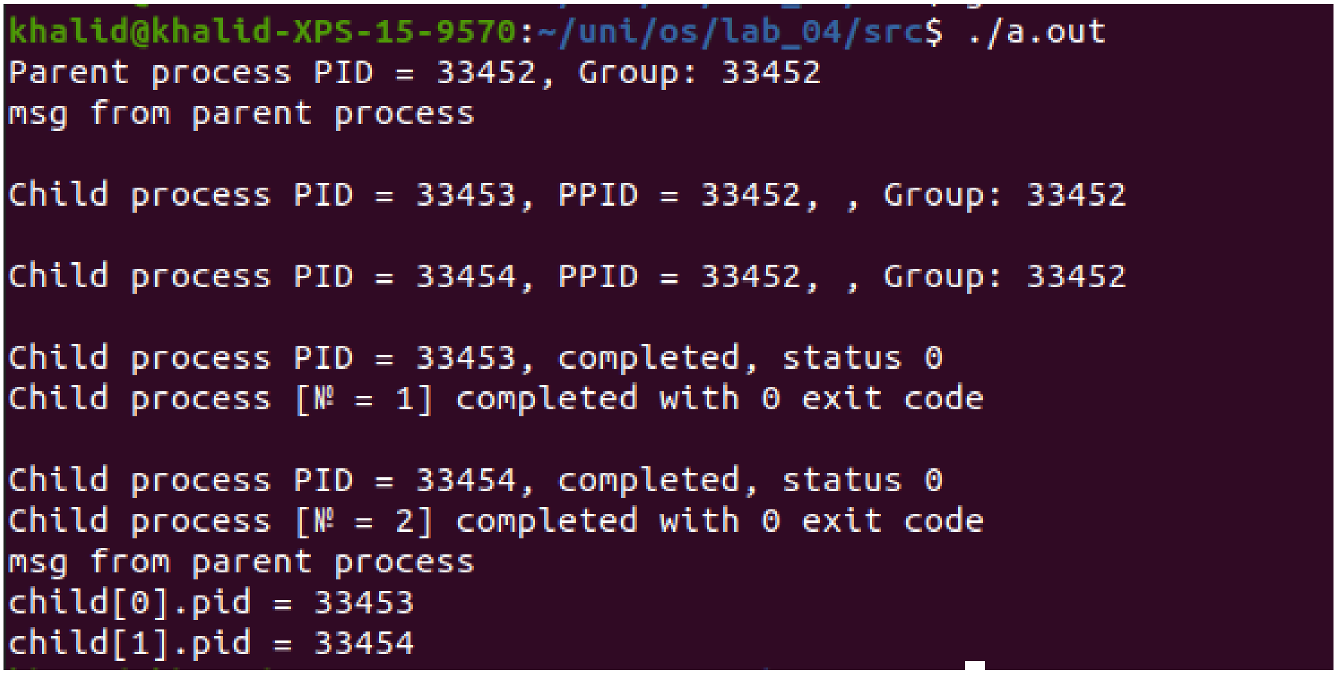
Рис. 1: Демонстрация работы программы (задание No1).

Задание No2

Предок ждет завершения своих потомком, используя системный вызов wait(). Вывод соответствующих сообщений на экран.

Листинг 2: Вызов функции wait()

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

#define REP_N 2 // loop rep time
#define SLP_T 2 // sleep time

enum error_code {
    ok,
    error
};

int main()
{
    int child[REP_N];


    printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());

    for (size_t i = 0; i < REP_N; i++) {
        pid_t pid = fork();

        if (pid == -1) {
            perror("Can't fork!");
            return error;
        } else if (pid == 0) {
            sleep(SLP_T);
            printf("\nChild process PID = %d, PPID = %d, , Group: %d\n", \
                                    getpid(), getppid(), getpgrp());
            return ok;
        } else {
            child[i] = pid;
        }

    }
    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++) {
        int status, ret_val = 0;

        pid_t child_pid = wait(&status);

        printf("\nChild process PID = %d, completed, status %d\n", \
                                    child_pid, status);

        if (WIFEXITED(ret_val))
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WEXITSTATUS(ret_val));
        else if (WIFSIGNALED(ret_val))
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WTERMSIG(ret_val));
        else if (WIFSTOPPED(ret_val))
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WSTOPSIG(ret_val));
    }

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++)
        printf("child[%zu].pid = %d\n", i, child[i]);

    return ok;
}
```



```
khalid@khalid-XPS-15-9570:~/uni/os/lab_04/src$ ./a.out
Parent process PID = 33452, Group: 33452
msg from parent process

Child process PID = 33453, PPID = 33452, , Group: 33452

Child process PID = 33454, PPID = 33452, , Group: 33452

Child process PID = 33453, completed, status 0
Child process [№ = 1] completed with 0 exit code

Child process PID = 33454, completed, status 0
Child process [№ = 2] completed with 0 exit code
msg from parent process
child[0].pid = 33453
child[1].pid = 33454
```

Рис. 2: Демонстрация работы программы (задание No2).

Задание No3

Потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 3: Вызов функции execvp()

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

#define REP_N 2 // loop rep time
#define SLP_T 2 // sleep time

enum error_code {
    ok,
    error
};

int main()
{
    int child[REP_N];
    char *const args[4] = { NULL };
    char *const s_args[4] = { NULL };
    int pid;

    printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());
```

```c

    for (size_t i = 0; i < REP_N; i++) {
        pid = fork();

        if (pid == -1) {
            perror("Fork failed!");
            return error;
        } else if (pid == 0) {
            printf("\nChild process PID = %d, PPID = %d, Group: %d\n", \
                        getpid(), getppid(), getpgrp());

            int res;
            if (i == 0)
                res = execvp("/home/khalid/Desktop/c_labs/rk_03/a.out", args);
            else
                res = execvp("/home/khalid/Desktop/c_labs/rk_04/a.out", args);

            if (res == -1) {
                perror("exec failed!");
                return error;
            }

            return ok;
        } else {
            wait(NULL);
            child[i] = pid;
        }
    }

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++) {
        int status, ret_val = 0;

        pid_t child_pid = wait(&status);

        printf("\nChild process PID = %d, completed, status %d\n", \
                                    child_pid, status);

        if (WIFEXITED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                       i + 1, WEXITSTATUS(ret_val));
        } else if (WIFSIGNALED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                       i + 1, WTERMSIG(ret_val));
        } else if (WIFSTOPPED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                       i + 1, WSTOPSIG(ret_val));
        }
    }

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++)
        printf("child[%zu].pid = %d\n", i, child[i]);

    return ok;
}
```

Листинг 4: код запушеного программа.

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

typedef struct
{
    int rows;
    int cols;
    int **mat;
} matrix;

enum error_code
{
    ok,
    error
};

matrix *create_matrix(const size_t rows, const size_t cols)
{
    matrix *tmp = malloc(sizeof(matrix));
    if (tmp)
    {
        tmp->rows = rows;
        tmp->cols = cols;
        tmp->mat = malloc(rows * sizeof(int *) + rows * cols * sizeof(int));
        if (tmp->mat)
        {
            for (int i = 0; i < rows; i++)
                tmp->mat[i] = (int *)((char *) tmp->mat + rows * sizeof(int
                    *) + i * cols * sizeof(int));
        }
        else
        {
            free(tmp);
            tmp = NULL;
        }
    }
    return tmp;
}

void display_mat(matrix *mat)
{
    for (int i = 0; i < mat->rows; i++)
    {
        for (int j = 0; j < mat->cols; j++)
            printf("%5d ", mat->mat[i][j]);
        printf("\n");
    }
}

int fill_mat(matrix *mat)
{
    int x;
    for (int i = 0; i < mat->rows; i++)
    {
        for (int j = 0; j < mat->cols; j++)
        {
            if (fscanf(stdin, "%d", &x) != 1)
                return error;
            mat->mat[i][j] = x;
        }
    }
    return ok;
}

int cmpint(const void *lhs, const void *rhs)
{
    return *(int *) lhs - *(int *) rhs;
}

void fill_mat_eles_with_arr(matrix *mat, int *arr)
{
    int i, k = 0, l = 0, d = mat->cols * mat->rows - 1;

    int m = mat->rows, n = mat->cols;

    while (k < m && l < n)
    {
        for (i = l; i < n; ++i)
        {
            mat->mat[k][i] = arr[d--];
        }
        k++;

        for (i = k; i < m; ++i)
        {
            mat->mat[i][n - 1] = arr[d--];
```

```c
        }
        n--;

        if (k < m)
        {
            for (i = n - 1; i >= l; --i)
            {
                mat->mat[m - 1][i] = arr[d--];
            }
            m--;
        }

        if (l < mat->cols)
        {
            for (i = m - 1; i >= k; --i)
            {
                mat->mat[i][l] = arr[d--];
            }
            l++;
        }
    }
}

void fill_arr_ele_with_mat(matrix *mat, int *arr)
{
    for (int i = 0; i < mat->rows; i++)
    {
        for (int j = 0; j < mat->cols; j++)
        {
            arr[i * mat->cols + j] = mat->mat[i][j];
        }
    }
}

void sort_snake_like_mat(matrix *mat)
{
    int *temp_mat = malloc(mat->rows * mat->cols * sizeof(int));
    if (temp_mat)
    {
        fill_arr_ele_with_mat(mat, temp_mat);
        qsort(temp_mat, mat->rows * mat->cols, sizeof(int), cmpint);
        fill_mat_eles_with_arr(mat, temp_mat);
        free(temp_mat);
    }
}

void free_mat(matrix *mat)
{
    free(mat->mat);
    free(mat);
```

```
}

int in_range(int x)
{
    return x > 0 && x < 100;
}

int main()
{
    int m, n, rc = ok;
    if (scanf("%d %d", &m, &n) == 2 && in_range(m) && in_range(n) && m == n)
    {
        matrix *mat = create_matrix(m, n);
        if (mat)
        {
            if (fill_mat(mat) == ok)
            {
                sort_snake_like_mat(mat);
                display_mat(mat);
            }
            else
                rc = error;

            free_mat(mat);
        }
        else
            rc = error;
    }
    else
        rc = error;
    return rc;
}
```

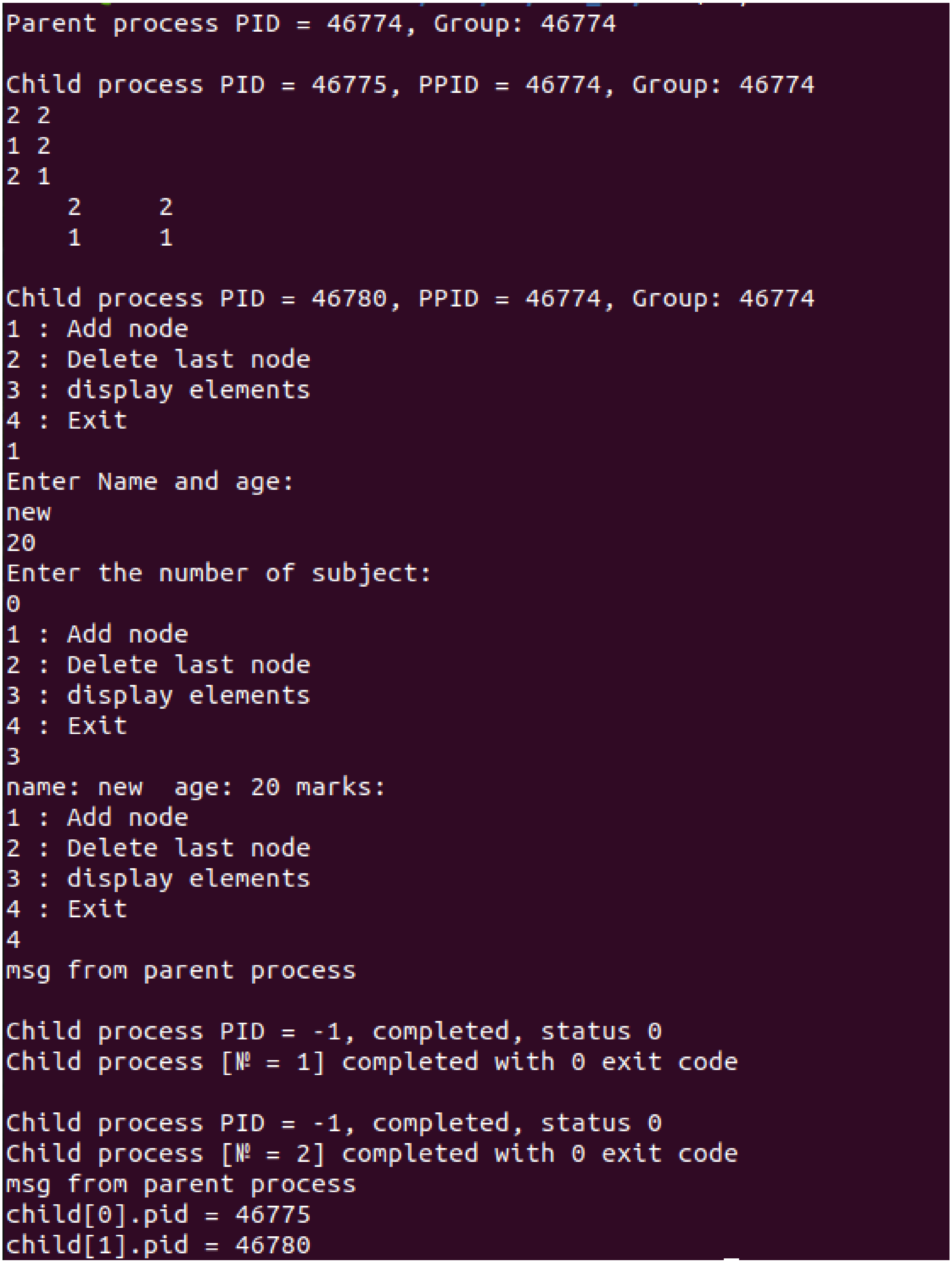
Листинг 5: код 2-ого запушеного программа.

```
#include "main.h"

node_t *create_ll_node(char *name, int age, node_t *marks)
{
    node_t *temp = calloc(1, sizeof(node_t));
    char *name_temp = NULL;
    if (temp)
    {
        if ((name_temp = strdup(name)) != NULL)
        {
            temp->name = name_temp;
            temp->age = age;
            temp->mark = marks;
        }
        else
```

```c
        {
            free(temp);
        }
    }
    return temp;
}

linked_list *create_ll(void)
{
    linked_list* temp = calloc(1, sizeof(linked_list));
    return temp;
}

int push_back(linked_list *list, node_t *new_node)
{
    if (new_node)
    {
        if (list->head == NULL)
        {
            list->head = list->end = new_node;
        }
        else
        {
            list->end->next = new_node;
            list->end = new_node;
        }
        return ok;
    }
    return error;
}

void pop_back(linked_list *list)
{
    node_t *temp, *n_temp = NULL;
    temp = list->head;
    while (temp->next)
    {
        n_temp = temp;
        temp = temp->next;
    }
    if (n_temp)
    {
        n_temp->next = NULL;
        if (temp->mark)
        {
            free(temp->name);
        }
        free(temp);
    }
    else
    {
        if (temp->mark)
        {
            free(temp->name);
        }
        free(temp);
        list->head = NULL;
    }
}

void display(linked_list *list)
{
    node_t *temp = list->head;
    node_t *marks = NULL;
    while (temp)
    {
        printf("name: %s  age: %d marks: ", temp->name, temp->age);
        marks = temp->mark;
        while (marks)
        {
            printf("subject: %s Numbers: %d ", marks->name, marks->age);
            marks = marks->next;
        }
        printf("\n");
        temp = temp->next;
    }
}

void free_ll(linked_list *list)
{
    node_t *temp;
    while (list->head)
    {
        temp = list->head;
        list->head = list->head->next;
        if (temp->mark)
        {
            free(temp->mark->name);
            free(temp->mark);
        }
        free(temp);
    }
    free(list);
}

char *get_str(void)
{
    size_t len;
    char *temp = NULL;
    ssize_t read = getline(&temp, &len, stdin);
    if (read > 0)
    {
        temp[read - 1] = '\0';
    }
    return temp;
}

void clear_stream(void)
{
    int x;
    while ((x = getchar()) != '\n' && x != EOF)
    {
    }
}

int main()
{
    linked_list *list = create_ll();
    char *name, *sub_name;
    int x, choice;
    if (list)
    {
        do
        {
            printf("1 : Add node\n2 : Delete last node\n3 : display elements\n4 : Exit\n");
            if (fscanf(stdin, "%d", &choice) == 1 && (choice < 1 || choice > 4))
            {
                puts("incorect choice: try again");
                break;
            }
            if (choice == 4)
            {
                break;
            }
            clear_stream();
            switch (choice)
            {
            case 1:
            {
                puts("Enter Name and age:");
                if ((name = get_str()) != NULL && fscanf(stdin, "%d", &x) == 1)
                {
                    if (push_back(list, create_ll_node(name, x, NULL)) == error)
                    {
                        puts("memeory allocation error: try again");
                    }
```

```c
                    else
                    {
                        linked_list *temp = create_ll();
                        if (temp)
                        {
                            puts("Enter the number of subject: ");
                            int num_of_subs;
                            if (fscanf(stdin, "%d", &num_of_subs) == 1 && num_of_subs > 0)
                            {
                                clear_stream();
                                for (int i = 0; i < num_of_subs; i++)
                                {
                                    printf("Sub no[%d]: name and mark\n", i + 1);
                                    // clear_stream();
                                    if ((sub_name = get_str()) != NULL && fscanf(stdin, "%d", &x) == 1)
                                    {
                                        if (push_back(temp, create_ll_node(sub_name, x, NULL)) == error)
                                        {
                                            puts("memeory allocation error: try again");
                                        }
                                        free(sub_name);
                                    }
                                    else
                                    {
                                        puts("Memory alloc error!");
                                        break;
                                    }
                                    clear_stream();
                                }
                            }
                            else if (num_of_subs != 0)
                            {
                                puts("incorrect input!");
                            }
                            list->end->mark = temp->head;
                            free(temp);
                        }
                        else
                        {
                            puts("memoy alloc error!");
                        }
                    }
                    free(name);
                }
                else
```

```
                    {
                        puts("Incorrect input or memeory Error");
                    }
                    break;
                }
                case 2:
                {
                    pop_back(list);
                    break;
                }
                case 3:
                {
                    display(list);
                    break;
                }
                }
            } while (choice);
            free_ll(list);
    }
}
```



```
Parent process PID = 46774, Group: 46774

Child process PID = 46775, PPID = 46774, Group: 46774
2 2
1 2
2 1
    2     2
    1     1

Child process PID = 46780, PPID = 46774, Group: 46774
1 : Add node
2 : Delete last node
3 : display elements
4 : Exit
1
Enter Name and age:
new
20
Enter the number of subject:
0
1 : Add node
2 : Delete last node
3 : display elements
4 : Exit
3
name: new  age: 20 marks:
1 : Add node
2 : Delete last node
3 : display elements
4 : Exit
4
msg from parent process

Child process PID = -1, completed, status 0
Child process [№ = 1] completed with 0 exit code

Child process PID = -1, completed, status 0
Child process [№ = 2] completed with 0 exit code
msg from parent process
child[0].pid = 46775
child[1].pid = 46780
```

Рис. 3: Демонстрация работы программы (задание No3).

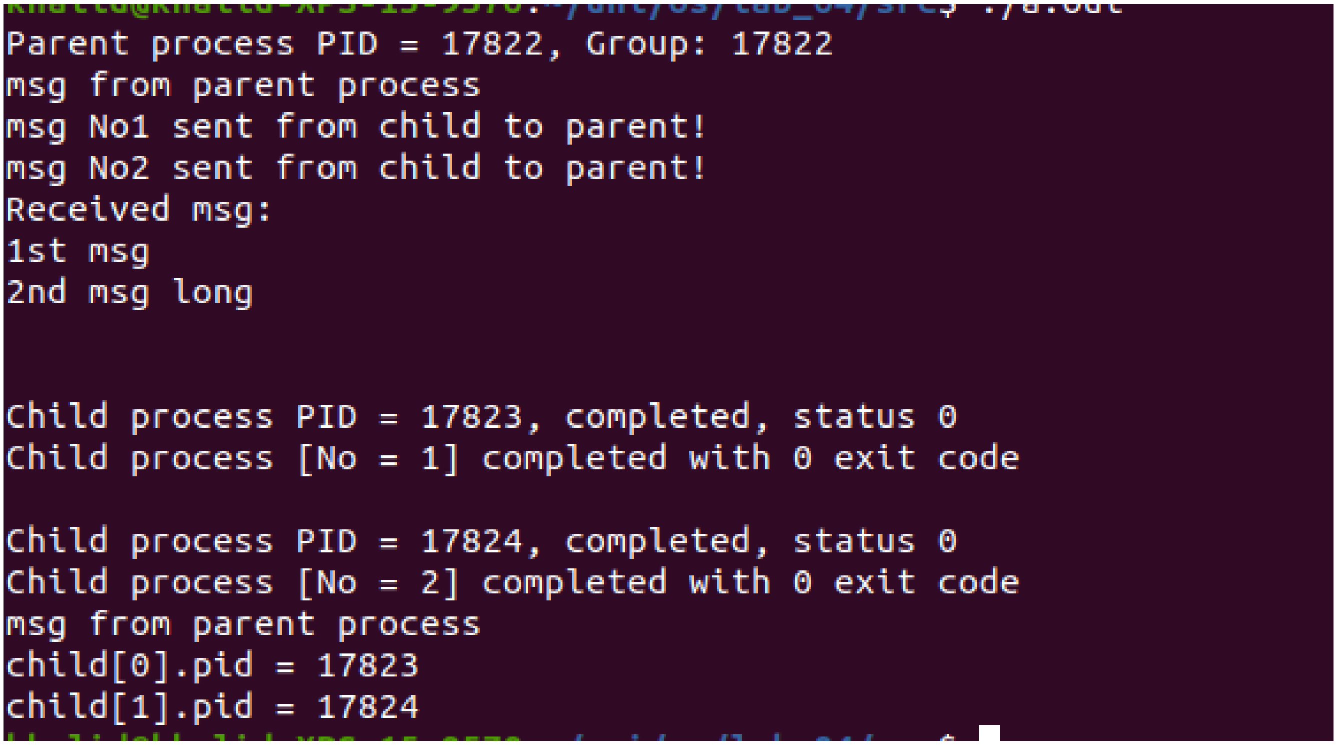
Задание No4

Предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 6: Использование pipe

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

#define REP_N 2 // loop rep time
#define SLP_T 2 // sleep time
#define BUF_SIZE (sizeof(char) * 256)

enum error_code {
    ok,
    pipe_error,
    exec_error,
    fork_error,
    error
};

void read_message(const int *fd, char *buff, const int _end, const int _begin) {
    close(fd[_end]);

    int index = 0;

    while (read(fd[_begin], &buff[index++], 1) != 0) { }

    buff[index] = '\0';
}

int main()
{
    int child[REP_N], fd[REP_N];
    char buffer[BUF_SIZE];
    char *const msg[REP_N] = {"1st msg\n", "2nd msg long\n"};
    int pid;

    if (REP_N < 2) {
        perror("at most needs to create two childs!");
        return error;
    }

    if (pipe(fd) == -1) {
        perror("can't pipe!");
        return pipe_error;
    }

    printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());

    for (size_t i = 0; i < REP_N; i++) {
        pid = fork();

        if (pid == -1) {
            perror("Fork failed!");
            return fork_error;
        } else if (pid == 0) {

            close(fd[0]);
            write(fd[1], msg[i], strlen(msg[i]));
            printf("msg No%zu sent from child to parent!\n", i + 1);

            return ok;
        } else {
            child[i] = pid;
        }
    }

    puts("msg from parent process");

    read_message(fd, buffer, 1, 0);
    printf("Received msg: \n%s\n", buffer);

    for (size_t i = 0; i < REP_N; i++) {
        int status, ret_val = 0;

        pid_t child_pid = wait(&status);

        printf("\nChild process PID = %d, completed, status %d\n", \
                                    child_pid, status);

        if (WIFEXITED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WEXITSTATUS(ret_val));
        } else if (WIFSIGNALED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WTERMSIG(ret_val));
        } else if (WIFSTOPPED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WSTOPSIG(ret_val));
        }
    }

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++)
```

```
        printf("child[%zu].pid = %d\n", i, child[i]);

    return ok;
}
```



```
Parent process PID = 17822, Group: 17822
msg from parent process
msg No1 sent from child to parent!
msg No2 sent from child to parent!
Received msg:
1st msg
2nd msg long


Child process PID = 17823, completed, status 0
Child process [No = 1] completed with 0 exit code

Child process PID = 17824, completed, status 0
Child process [No = 2] completed with 0 exit code
msg from parent process
child[0].pid = 17823
child[1].pid = 17824
```

Рис. 4: Демонстрация работы программы (задание No4).

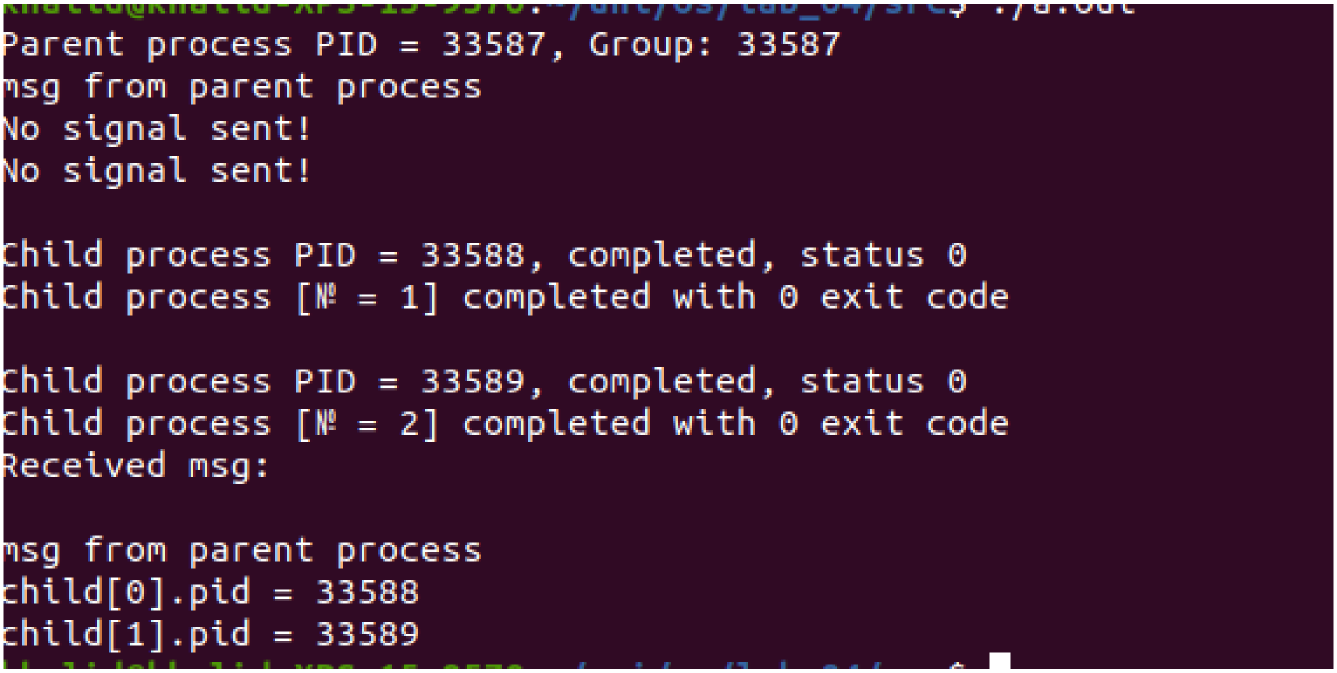
Задание No5

Предок и потомки обмениваются сообщениями через неименованный программный канал. С помощью сигнала меняется ход выполнения программы. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг 7: Использование сигналов

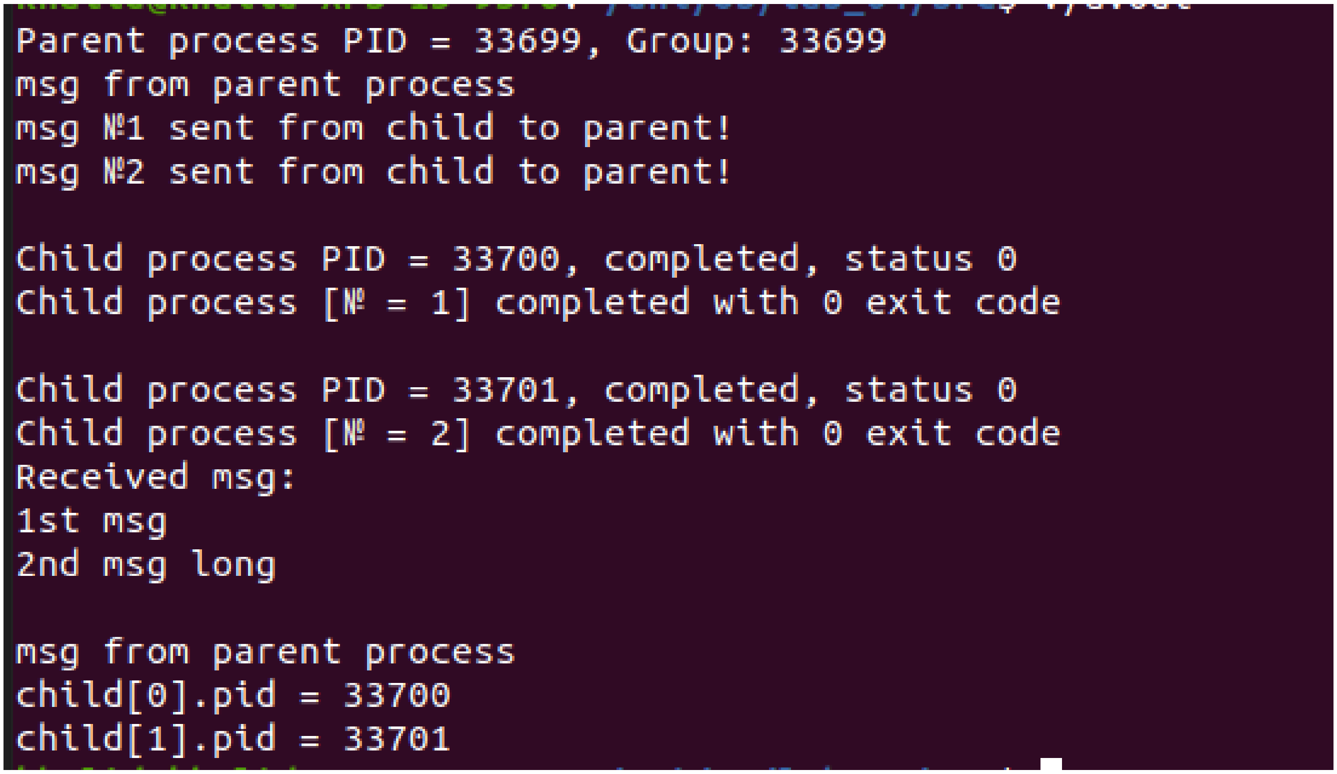
```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>

#define REP_N 2 // loop rep time
#define SLP_T 2 // sleep time
#define BUF_SIZE (sizeof(char) * 256)
#define GET 1

enum error_code {
    ok,
    pipe_error,
    exec_error,
    fork_error,
    error
};

int mode = 0;

void sigint_catcher(int signum) {
    printf( "\nProccess Catched signal #%d\n", signum);
    printf("Sent any message to clihd!\n");
    mode = 1;
}
int main()
{
    int child[REP_N], fd[REP_N];
    char buffer[BUF_SIZE] = { 0 };
    char *const msg[REP_N] = {"1st msg\n", "2nd msg long\n"};
    int pid;

    if (REP_N < 2) {
        perror("at most needs to create two childs!");
        return error;
    }

    if (pipe(fd) == -1) {
        perror("can't pipe!");
        return pipe_error;
    }

    printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());
    signal(SIGINT, sigint_catcher);

    for (size_t i = 0; i < REP_N; i++) {
        pid = fork();

        if (pid == -1) {
            perror("Fork failed!");
            return fork_error;
        } else if (pid == 0) {
            signal(SIGINT, sigint_catcher);

            if (mode) {
                close(fd[0]);
                write(fd[1], msg[i], strlen(msg[i]));
                printf("msg No%zu sent from child to parent!\n", i + 1);
            } else {
                printf("No signal sent!\n");
            }

```

```
                return ok;
            } else {
                child[i] = pid;
            }
    }

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++) {
        int status, ret_val = 0;

        pid_t child_pid = wait(&status);

        printf("\nChild process PID = %d, completed, status %d\n", \
                                    child_pid, status);

        if (WIFEXITED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WEXITSTATUS(ret_val));
        } else if (WIFSIGNALED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WTERMSIG(ret_val));
        } else if (WIFSTOPPED(ret_val)) {
            printf("Child process [No = %zu] completed with %d exit code\n",
                                        i + 1, WSTOPSIG(ret_val));
        }
    }

    close(fd[1]);
    read(fd[0], buffer, BUF_SIZE);
    printf("Received msg: \n%s\n", buffer);

    puts("msg from parent process");

    for (size_t i = 0; i < REP_N; i++)
        printf("child[%zu].pid = %d\n", i, child[i]);

    return ok;
}
```



```
Parent process PID = 33587, Group: 33587
msg from parent process
No signal sent!
No signal sent!

Child process PID = 33588, completed, status 0
Child process [№ = 1] completed with 0 exit code

Child process PID = 33589, completed, status 0
Child process [№ = 2] completed with 0 exit code
Received msg:

msg from parent process
child[0].pid = 33588
child[1].pid = 33589
```

Рис. 5: Демонстрация работы программы, сигнал не вызывается (задание No5).



```
Parent process PID = 33699, Group: 33699
msg from parent process
msg №1 sent from child to parent!
msg №2 sent from child to parent!

Child process PID = 33700, completed, status 0
Child process [№ = 1] completed with 0 exit code

Child process PID = 33701, completed, status 0
Child process [№ = 2] completed with 0 exit code
Received msg:
1st msg
2nd msg long

msg from parent process
child[0].pid = 33700
child[1].pid = 33701
```

Рис. 6: Демонстрация работы программы, сигнал вызывается (задание No5).