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;
```

```
27          } else if (pid == 0) {
28              sleep(SLP_T);
29              printf("\nChild process PID = %d, PPID = %d, , Group: %d\n", \
30                                      getpid(), getppid(), getpgrp());
31              return ok;
32          } else {
33              child[i] = pid;
34          }
35
36      }
37      puts("msg from parent process");
38
39      for (size_t i = 0; i < REP_N; i++) {
40          int status, ret_val = 0;
41
42          pid_t child_pid = wait(&status);
43
44          printf("\nChild process PID = %d, completed, status %d\n", \
45                                      child_pid, status);
46
47          if (WIFEXITED(ret_val))
48              printf("Child process [No = %zu] completed with %d exit code\n",
49                                          i + 1, WEXITSTATUS(ret_val));
50          else if (WIFSIGNALED(ret_val))
51              printf("Child process [No = %zu] completed with %d exit code\n",
52                                          i + 1, WTERMSIG(ret_val));
53          else if (WIFSTOPPED(ret_val))
54              printf("Child process [No = %zu] completed with %d exit code\n",
55                                          i + 1, WSTOPSIG(ret_val));
56      }
57
58      puts("msg from parent process");
59
60      for (size_t i = 0; i < REP_N; i++)
61          printf("child[%zu].pid = %d\n", i, child[i]);
62
63      return ok;
64  }
```

```
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()

```c
#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());
```

```
22
23      for (size_t i = 0; i < REP_N; i++) {
24          pid = fork();
25
26          if (pid == -1) {
27              perror("Fork failed!");
28              return error;
29          } else if (pid == 0) {
30              printf("\nChild process PID = %d, PPID = %d, Group: %d\n", \
31                          getpid(), getppid(), getpgrp());
32
33              int res;
34              if (i == 0)
35                  res = execvp("/home/khalid/Desktop/c_labs/rk_03/a.out", args
                        );
36              else
37                  res = execvp("/home/khalid/Desktop/c_labs/rk_04/a.out", args
                        );
38
39              if (res == -1) {
40                  perror("exec failed!");
41                  return error;
42              }
43
44              return ok;
45          } else {
46              wait(NULL);
47              child[i] = pid;
48          }
49      }
50
51      puts("msg from parent process");
52
53      for (size_t i = 0; i < REP_N; i++) {
54          int status, ret_val = 0;
55
56          pid_t child_pid = wait(&status);
57
58          printf("\nChild process PID = %d, completed, status %d\n", \
59                                      child_pid, status);
60
61          if (WIFEXITED(ret_val)) {
62              printf("Child process [No = %zu] completed with %d exit code\n",
63                                          i + 1, WEXITSTATUS(ret_val));
64          } else if (WIFSIGNALED(ret_val)) {
65              printf("Child process [No = %zu] completed with %d exit code\n",
66                                          i + 1, WTERMSIG(ret_val));
67          } else if (WIFSTOPPED(ret_val)) {
68              printf("Child process [No = %zu] completed with %d exit code\n",
69                                          i + 1, WSTOPSIG(ret_val));
```

```
70          }
71      }
72
73      puts("msg from parent process");
74
75      for (size_t i = 0; i < REP_N; i++)
76          printf("child[%zu].pid = %d\n", i, child[i]);
77
78      return ok;
79  }
```

Листинг 4: код запушеного программа.

```
1   #include <stdio.h>
2   #include <math.h>
3   #include <stdlib.h>
4
5   typedef struct
6   {
7       int rows;
8       int cols;
9       int **mat;
10  } matrix;
11
12  enum error_code
13  {
14      ok,
15      error
16  };
17
18  matrix *create_matrix(const size_t rows, const size_t cols)
19  {
20      matrix *tmp = malloc(sizeof(matrix));
21      if (tmp)
22      {
23          tmp->rows = rows;
24          tmp->cols = cols;
25          tmp->mat = malloc(rows * sizeof(int *) + rows * cols * sizeof(int));
26          if (tmp->mat)
27          {
28              for (int i = 0; i < rows; i++)
29                  tmp->mat[i] = (int *)((char *) tmp->mat + rows * sizeof(int
                        *) + i * cols * sizeof(int));
30          }
31          else
32          {
33              free(tmp);
34              tmp = NULL;
35          }
36      }
```

```
37      return tmp;
38  }
39
40  void display_mat(matrix *mat)
41  {
42      for (int i = 0; i < mat->rows; i++)
43      {
44          for (int j = 0; j < mat->cols; j++)
45              printf("%5d ", mat->mat[i][j]);
46          printf("\n");
47      }
48  }
49
50  int fill_mat(matrix *mat)
51  {
52      int x;
53      for (int i = 0; i < mat->rows; i++)
54      {
55          for (int j = 0; j < mat->cols; j++)
56          {
57              if (fscanf(stdin, "%d", &x) != 1)
58                  return error;
59              mat->mat[i][j] = x;
60          }
61      }
62      return ok;
63  }
64
65  int cmpint(const void *lhs, const void *rhs)
66  {
67      return *(int *) lhs - *(int *) rhs;
68  }
69
70  void fill_mat_eles_with_arr(matrix *mat, int *arr)
71  {
72      int i, k = 0, l = 0, d = mat->cols * mat->rows - 1;
73
74      int m = mat->rows, n = mat->cols;
75
76      while (k < m && l < n)
77      {
78          for (i = l; i < n; ++i)
79          {
80              mat->mat[k][i] = arr[d--];
81          }
82          k++;
83
84          for (i = k; i < m; ++i)
85          {
86              mat->mat[i][n - 1] = arr[d--];
```

```
87          }
88          n--;
89
90          if (k < m)
91          {
92              for (i = n - 1; i >= l; --i)
93              {
94                  mat->mat[m - 1][i] = arr[d--];
95              }
96              m--;
97          }
98
99          if (l < mat->cols)
100         {
101             for (i = m - 1; i >= k; --i)
102             {
103                 mat->mat[i][l] = arr[d--];
104             }
105             l++;
106         }
107     }
108 }
109
110 void fill_arr_ele_with_mat(matrix *mat, int *arr)
111 {
112     for (int i = 0; i < mat->rows; i++)
113     {
114         for (int j = 0; j < mat->cols; j++)
115         {
116             arr[i * mat->cols + j] = mat->mat[i][j];
117         }
118     }
119 }
120
121 void sort_snake_like_mat(matrix *mat)
122 {
123     int *temp_mat = malloc(mat->rows * mat->cols * sizeof(int));
124     if (temp_mat)
125     {
126         fill_arr_ele_with_mat(mat, temp_mat);
127         qsort(temp_mat, mat->rows * mat->cols, sizeof(int), cmpint);
128         fill_mat_eles_with_arr(mat, temp_mat);
129         free(temp_mat);
130     }
131 }
132
133 void free_mat(matrix *mat)
134 {
135     free(mat->mat);
136     free(mat);
```

```
137 }
138
139 int in_range(int x)
140 {
141     return x > 0 && x < 100;
142 }
143
144 int main()
145 {
146     int m, n, rc = ok;
147     if (scanf("%d %d", &m, &n) == 2 && in_range(m) && in_range(n) && m == n)
148     {
149         matrix *mat = create_matrix(m, n);
150         if (mat)
151         {
152             if (fill_mat(mat) == ok)
153             {
154                 sort_snake_like_mat(mat);
155                 display_mat(mat);
156             }
157             else
158                 rc = error;
159
160             free_mat(mat);
161         }
162         else
163             rc = error;
164     }
165     else
166         rc = error;
167     return rc;
168 }
```

Листинг 5: код 2-ого запушеного программа.

```
1 #include "main.h"
2
3 node_t *create_ll_node(char *name, int age, node_t *marks)
4 {
5     node_t *temp = calloc(1, sizeof(node_t));
6     char *name_temp = NULL;
7     if (temp)
8     {
9         if ((name_temp = strdup(name)) != NULL)
10        {
11            temp->name = name_temp;
12            temp->age = age;
13            temp->mark = marks;
14        }
15        else
```

```
16          {
17              free(temp);
18          }
19      }
20      return temp;
21  }
22
23  linked_list *create_ll(void)
24  {
25      linked_list* temp = calloc(1, sizeof(linked_list));
26      return temp;
27  }
28
29  int push_back(linked_list *list, node_t *new_node)
30  {
31      if (new_node)
32      {
33          if (list->head == NULL)
34          {
35              list->head = list->end = new_node;
36          }
37          else
38          {
39              list->end->next = new_node;
40              list->end = new_node;
41          }
42          return ok;
43      }
44      return error;
45  }
46
47  void pop_back(linked_list *list)
48  {
49      node_t *temp, *n_temp = NULL;
50      temp = list->head;
51      while (temp->next)
52      {
53          n_temp = temp;
54          temp = temp->next;
55      }
56      if (n_temp)
57      {
58          n_temp->next = NULL;
59          if (temp->mark)
60          {
61              free(temp->name);
62          }
63          free(temp);
64      }
65      else
```

```
66      {
67          if (temp->mark)
68          {
69              free(temp->name);
70          }
71          free(temp);
72          list->head = NULL;
73      }
74  }
75
76  void display(linked_list *list)
77  {
78      node_t *temp = list->head;
79      node_t *marks = NULL;
80      while (temp)
81      {
82          printf("name: %s  age: %d marks: ", temp->name, temp->age);
83          marks = temp->mark;
84          while (marks)
85          {
86              printf("subject: %s Numbers: %d ", marks->name, marks->age);
87              marks = marks->next;
88          }
89          printf("\n");
90          temp = temp->next;
91      }
92  }
93
94  void free_ll(linked_list *list)
95  {
96      node_t *temp;
97      while (list->head)
98      {
99          temp = list->head;
100         list->head = list->head->next;
101         if (temp->mark)
102         {
103             free(temp->mark->name);
104             free(temp->mark);
105         }
106         free(temp);
107     }
108     free(list);
109 }
110
111 char *get_str(void)
112 {
113     size_t len;
114     char *temp = NULL;
115     ssize_t read = getline(&temp, &len, stdin);
```

```
116     if (read > 0)
117     {
118         temp[read - 1] = '\0';
119     }
120     return temp;
121 }
122 
123 void clear_stream(void)
124 {
125     int x;
126     while ((x = getchar()) != '\n' && x != EOF)
127     {
128     }
129 }
130 
131 int main()
132 {
133     linked_list *list = create_ll();
134     char *name, *sub_name;
135     int x, choice;
136     if (list)
137     {
138         do
139         {
140             printf("1 : Add node\n2 : Delete last node\n3 : display elements
                \n4 : Exit\n");
141             if (fscanf(stdin, "%d", &choice) == 1 && (choice < 1 || choice >
                 4))
142             {
143                 puts("incorect choice: try again");
144                 break;
145             }
146             if (choice == 4)
147             {
148                 break;
149             }
150             clear_stream();
151             switch (choice)
152             {
153             case 1:
154             {
155                 puts("Enter Name and age:");
156                 if ((name = get_str()) != NULL && fscanf(stdin, "%d", &x) ==
                     1)
157                 {
158                     if (push_back(list, create_ll_node(name, x, NULL)) ==
                        error)
159                     {
160                         puts("memeory allocation error: try again");
161                     }
```

```
162                     else
163                     {
164                         linked_list *temp = create_ll();
165                         if (temp)
166                         {
167                             puts("Enter the number of subject: ");
168                             int num_of_subs;
169                             if (fscanf(stdin, "%d", &num_of_subs) == 1 &&
                                    num_of_subs > 0)
170                             {
171                                 clear_stream();
172                                 for (int i = 0; i < num_of_subs; i++)
173                                 {
174                                     printf("Sub no[%d]: name and mark\n", i
                                            + 1);
175                                     // clear_stream();
176                                     if ((sub_name = get_str()) != NULL &&
                                            fscanf(stdin, "%d", &x) == 1)
177                                     {
178                                         if (push_back(temp, create_ll_node(
                                                sub_name, x, NULL)) == error)
179                                         {
180                                             puts("memeory allocation error:
                                                    try again");
181                                         }
182                                         free(sub_name);
183                                     }
184                                     else
185                                     {
186                                         puts("Memory alloc error!");
187                                         break;
188                                     }
189                                     clear_stream();
190                                 }
191                             }
192                             else if (num_of_subs != 0)
193                             {
194                                 puts("incorrect input!");
195                             }
196                             list->end->mark = temp->head;
197                             free(temp);
198                         }
199                         else
200                         {
201                             puts("memoy alloc error!");
202                         }
203                     }
204                     free(name);
205                 }
206                 else
```

```
207                 {
208                     puts("Incorrect input or memeory Error");
209                 }
210                 break;
211             }
212             case 2:
213             {
214                 pop_back(list);
215                 break;
216             }
217             case 3:
218             {
219                 display(list);
220                 break;
221             }
222             }
223         } while (choice);
224         free_ll(list);
225     }
226 }
```

```
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

```
#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;
```

```
44      }
45
46      printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());
47
48      for (size_t i = 0; i < REP_N; i++) {
49          pid = fork();
50
51          if (pid == -1) {
52              perror("Fork failed!");
53              return fork_error;
54          } else if (pid == 0) {
55
56              close(fd[0]);
57              write(fd[1], msg[i], strlen(msg[i]));
58              printf("msg No%zu sent from child to parent!\n", i + 1);
59
60              return ok;
61          } else {
62              child[i] = pid;
63          }
64      }
65
66      puts("msg from parent process");
67
68      read_message(fd, buffer, 1, 0);
69      printf("Received msg: \n%s\n", buffer);
70
71      for (size_t i = 0; i < REP_N; i++) {
72          int status, ret_val = 0;
73
74          pid_t child_pid = wait(&status);
75
76          printf("\nChild process PID = %d, completed, status %d\n", \
77                                      child_pid, status);
78
79          if (WIFEXITED(ret_val)) {
80              printf("Child process [No = %zu] completed with %d exit code\n",
81                                          i + 1, WEXITSTATUS(ret_val));
82          } else if (WIFSIGNALED(ret_val)) {
83              printf("Child process [No = %zu] completed with %d exit code\n",
84                                          i + 1, WTERMSIG(ret_val));
85          } else if (WIFSTOPPED(ret_val)) {
86              printf("Child process [No = %zu] completed with %d exit code\n",
87                                          i + 1, WSTOPSIG(ret_val));
88          }
89      }
90
91      puts("msg from parent process");
92
93      for (size_t i = 0; i < REP_N; i++)
```

```
94          printf("child[%zu].pid = %d\n", i, child[i]);
95
96      return ok;
97  }
```

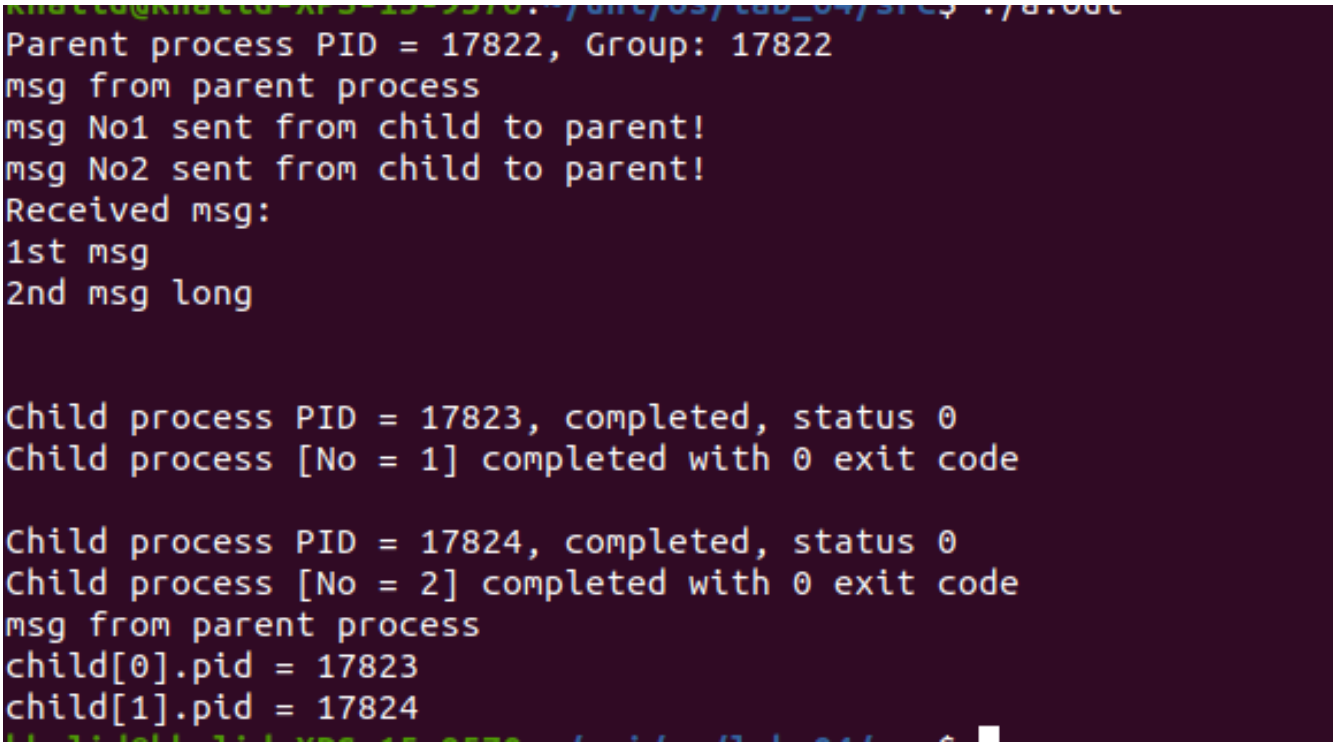

Рис. 4: Демонстрация работы программы (задание No4).

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

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <sys/wait.h>
5  #include <sys/types.h>
6
7  #define REP_N 2 // loop rep time
8  #define SLP_T 2 // sleep time
9  #define BUF_SIZE (sizeof(char) * 256)
10 #define GET 1
11
12 enum error_code {
13     ok,
```

```
14      pipe_error,
15      exec_error,
16      fork_error,
17      error
18  };
19
20  int mode = 0;
21
22  void sigint_catcher(int signum) {
23      printf( "\nProccess Catched signal #%d\n", signum);
24      printf("Sent any message to clihd!\n");
25      mode = 1;
26  }
27  int main()
28  {
29      int child[REP_N], fd[REP_N];
30      char buffer[BUF_SIZE] = { 0 };
31      char *const msg[REP_N] = {"1st msg\n", "2nd msg long\n"};
32      int pid;
33
34      if (REP_N < 2) {
35          perror("at most needs to create two childs!");
36          return error;
37      }
38
39      if (pipe(fd) == -1) {
40          perror("can't pipe!");
41          return pipe_error;
42      }
43
44      printf("Parent process PID = %d, Group: %d\n", getpid(), getpgrp());
45      signal(SIGINT, sigint_catcher);
46
47      for (size_t i = 0; i < REP_N; i++) {
48          pid = fork();
49
50          if (pid == -1) {
51              perror("Fork failed!");
52              return fork_error;
53          } else if (pid == 0) {
54              signal(SIGINT, sigint_catcher);
55
56              if (mode) {
57                  close(fd[0]);
58                  write(fd[1], msg[i], strlen(msg[i]));
59                  printf("msg No%zu sent from child to parent!\n", i + 1);
60              } else {
61                  printf("No signal sent!\n");
62              }
63
```

```
64              return ok;
65          } else {
66              child[i] = pid;
67          }
68      }
69
70      puts("msg from parent process");
71
72      for (size_t i = 0; i < REP_N; i++) {
73          int status, ret_val = 0;
74
75          pid_t child_pid = wait(&status);
76
77          printf("\nChild process PID = %d, completed, status %d\n", \
78                                      child_pid, status);
79
80          if (WIFEXITED(ret_val)) {
81              printf("Child process [No = %zu] completed with %d exit code\n",
82                                          i + 1, WEXITSTATUS(ret_val));
83          } else if (WIFSIGNALED(ret_val)) {
84              printf("Child process [No = %zu] completed with %d exit code\n",
85                                          i + 1, WTERMSIG(ret_val));
86          } else if (WIFSTOPPED(ret_val)) {
87              printf("Child process [No = %zu] completed with %d exit code\n",
88                                          i + 1, WSTOPSIG(ret_val));
89          }
90      }
91
92      close(fd[1]);
93      read(fd[0], buffer, BUF_SIZE);
94      printf("Received msg: \n%s\n", buffer);
95
96      puts("msg from parent process");
97
98      for (size_t i = 0; i < REP_N; i++)
99          printf("child[%zu].pid = %d\n", i, child[i]);
100
101     return ok;
102 }
```

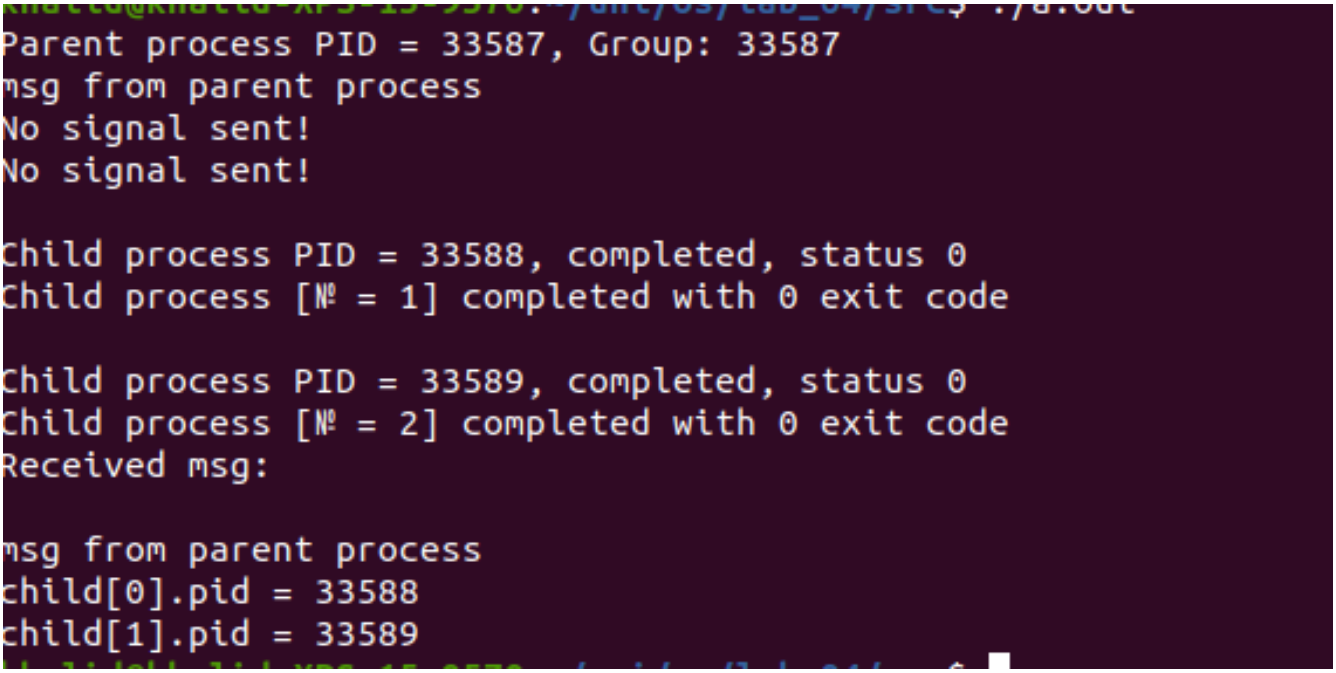

Рис. 5: Демонстрация работы программы, сигнал не вызывается (задание No5).

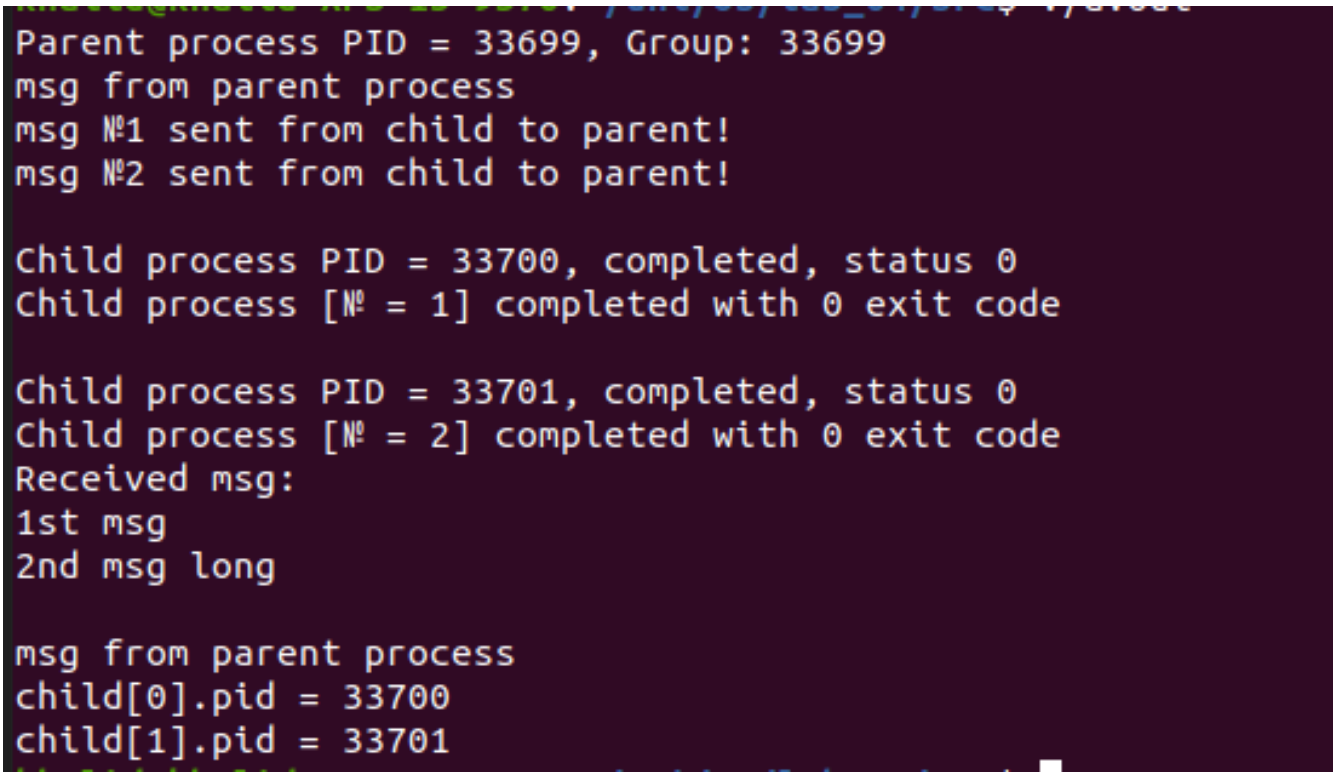

Рис. 6: Демонстрация работы программы, сигнал вызывается (задание No5).