

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»
КАФЕЛРА «Г	Грограммное 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)
Преподава	атель	 (Подпись, дата)	H. Ю. Рязанова (И. О. Фамилия)

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 ехес	18

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_{\square}born:_{\square}PID_{\square}=_{\square}%d;_{\square}PPID_{\square}=_{\square}%d;_{\square}GROUP_{\square}=_{\square}%d \setminus n",
           getpid(), getppid(), getpgrp());
  for (unsigned i = 0; i < CHILD_CNT; ++i)</pre>
     pid_t pid = fork();
     if (pid == -1)
     {
       fprintf(stderr, "Can't tork \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

На рисунке 1.1 приведен пример работы программы.

```
migregalqDESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_01.c
migregalqDESKTOP-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
migregalqDESKTOP-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

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_{\sqcup\sqcup}born: PID_{\sqcup} = _{\sqcup}\%d;_{\sqcup}PPID_{\sqcup} = _{\sqcup}\%d;_{\sqcup}PGRP_{\sqcup} = _{\sqcup}\%d \setminus n",
          getpid(), getppid(), getpgrp());
  for (unsigned i = 0; i < CHILD_CNT; ++i)</pre>
    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_%uudied:PIDu=u%d;uPPIDu=u%d;uPGRPu=u%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)</pre>
                     int status, stat_val = 0;
                     printf("parent<sub>□□</sub>waiting\n");
                     pid_t childpid = wait(&status);
                     printf("parent_uwaited:child_process_(PID_=\%d)_finished.
                                    status: ⊔%d\n",
                                                          childpid, status);
                     if (WIFEXITED(stat_val))
                     {
                                printf("parentultalk:childuprocessu#%dufinisheduwithucode:u
                                              %d\n", i + 1,
                                                                     WEXITSTATUS(stat_val));
                     else if (WIFSIGNALED(stat_val))
                     {
                                printf("parentuutalk:childuprocessu#%dufinishedubyusignalu
                                              with \( \code : \( \lambda \lambda \n \),
                                                                     i + 1, WTERMSIG(stat_val));
                     }
                     else if (WIFSTOPPED(stat_val))
                                printf("parent_{\sqcup\sqcup}talk: child_{\sqcup}process_{\sqcup}\#\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}\%d_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}stopped_{\sqcup}with_{\sqcup}st
                                              d\n",
                                                                     i + 1, WSTOPSIG(stat_val));
                     }
           }
           printf("parent_\sqcup_died:PID_{\sqcup}=_{\sqcup}%d;_{\sqcup}PPID_{\sqcup}=_{\sqcup}%d;_{\sqcup}PGRP_{\sqcup}=_{\sqcup}%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

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_= $\lfloor \%d; \rfloor$ PPID $_{\perp}=$ $\lfloor \%d; \rfloor$ GROUP $_{\perp}=$ $\lfloor \%d \setminus n \rfloor$, getpid(), getppid(), getpgrp()); for (unsigned i = 0; i < CHILD_CNT; ++i)</pre> 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_{\sqcup}born:PID_{\sqcup}=_{\sqcup}%d;_{\sqcup}PPID_{\sqcup}=_{\sqcup}%d;_{\sqcup}GROUP_{\sqcup}=_{\sqcup}%d \setminus n",
                i, getpid(), 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;_"
             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 utalk: child process #%du"
                "finished \sqcup by \sqcup signal \sqcup with \sqcup code: \sqcup %d\n",
                i + 1, WTERMSIG(stat_val));
     }
     else if (WIFSTOPPED(stat_val))
       printf("parent_talk:child_process_#%d_"
                "finished_{\sqcup}stopped_{\sqcup}with_{\sqcup}code:_{\sqcup}%d_{\square},
                i + 1, WSTOPSIG(stat_val));
     }
  }
  printf("parent_died:PID_=\lfloor \%d; \rfloorPPID_{\perp}= \lfloor \%d; \rfloorGROUP_{\perp}= \lfloor \%d \setminus n \rfloor,
           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
Total files:
                           9
Real time spent:
                           16562000
Total time spent:
                           32
Avg filewarker time:
                           Θ
Avg digester queue time: 0.3333333333333333
Avg digester time:
                           3.22222222222223
Avg queue time:
53986243c8b4062afc0b4847aaaa5ae6 a.out
757c0df7c6a2aed08b9ac57562489296 data/data.txt
448de48ff7023508b390f29fc3b068f5 lab_05
a533bb519a9483e38b9744b5449dfece 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
                Time
           8.7323ms
   3|
           20.5318ms
   41
           35.377ms
   5
          45.7493ms
   6
           64.0028ms
   7
         100.8108ms
   8
          180.328ms
   9
         198.5162ms
  10|
          197.2043ms
          185.7465ms
  11|
     BRUTE ALGORITHM
  N
                Time
               5.3µs
   21
   3
              6.1µs
   4
              13.7µs|
   5
             82.5µs
   6 l
            352.9µs
   71
           2.0128ms
   8
          20.6795ms
   9
         218.5122ms
  10|
         1.5567464s
  111
        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

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\sqcupI\sqcupmaybe\sqcupyou\setminusn",
       "The \square autumn \square leaves \square of \square red \square and \square gold \square ;
  if (pipe(fd) == -1)
     fprintf(stderr, "Can't pipe \n");
     exit(ERR_PIPE);
  printf("parent_born:PID_=\lfloor \%d ; \lfloor PPID_{\perp} = \lfloor \%d ; \rfloor GROUP_{\perp} = \lfloor \%d \setminus n \rfloor,
```

```
Листинг 4.2 – Листинг main.c. Часть 2
          getpid(), getppid(), getpgrp());
  for (unsigned i = 0; i < CHILD_CNT; ++i)</pre>
    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_{\sqcup}born:PID_{\sqcup}=_{\sqcup}%d;_{\sqcup}PPID_{\sqcup}=_{\sqcup}%d;_{\sqcup}GROUP_{\sqcup}=_{\sqcup}%d \setminus 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;_"
             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 \lfloor by \rfloor signal \lfloor with \rfloor code: \lfloor \frac{n}{d} \rfloor,
               i + 1, WTERMSIG(stat_val));
```

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 recv:
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

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 O
#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<sub>□</sub>I<sub>□</sub>maybe<sub>□</sub>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'tupipe\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 tork \n");
      exit(ERR_FORK);
    if (pid)
    {
       child_pids[i] = pid;
    }
    else
    {
      printf("child_%u_{\sqcup}born:PID_{\sqcup}=_{\sqcup}%d;_{\sqcup}PPID_{\sqcup}=_{\sqcup}%d;_{\sqcup}GROUP_{\sqcup}=_{\sqcup}%d\backslash 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_%uusend:quietumodeunotumessageusend\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;_"
            childpid, status);
    if (WIFEXITED(stat_val))
      printf("parent_{\sqcup}talk:child_{\sqcup}process_{\sqcup}\#\%d_{\sqcup}"
              "finished \sqcup with \sqcup code: \sqcup %d\n",
              i + 1, WEXITSTATUS(stat_val));
    else if (WIFSIGNALED(stat_val))
      printf("parent_{\sqcup}talk: child_{\sqcup}process_{\sqcup}#%d_{\sqcup}"
              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("parentudied:PIDu=u%d;uPPIDu=u%d;uGROUPu=u%d\n",
         getpid(), getppid(), getpgrp());
}
```

На рисунке 5.1 приведен пример работы программы без вызова сигнала.

```
migregalaDESKTOP-INLN625:~/projects/bmstu_iu7_os/sem_01/lab_04/src$ gcc task_05.c
migregalaDESKTOP-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_1 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 talk:child process #2 finished with code: 0
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 ехес

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

			Mipaudh	r, 647-536
tork:	*	<i>i</i> 1	AN DIET A STELL M	nouncece hospieus
s Pegep	bupyetae npooppui	upoyeen ged	ganner a crews w	
2. Vazu	uraneu ugentugatue	solp 141) u aspa	usypa proc unoceres	3 From Espyulyper
3. Hang	oful of mayerca	woughered: ugent-	usypa proc notoelles Nemotopale nome por nouszobasent u nome unnymann mun gut noranna	Jupanes, BO.
ya ett	curve cos a gigan	a mongeties energy converse	unu ger normana	quarenewells.
PIO n	House e ero page	seure, guazanteux	na copyright proc	populant.
4. Cozgai	after napolal sporueren	enjuer agrecol ger	upeyeecs notionens.	
5. Buga	weemer odvace u		ue vompyenue d	eacts u
	eca-upisua.			
6. Uzu-c	d economic odeacte	ig us uphone un	robe agrecayum u u	p-lo donumera.
7. Heroun	on godstancial &	uadop upoyeccob,	note agrecayum u u note passement nese.	ode-18 megae
& Detac	Some of all	igeria poja e		
agreeny	un nosouina.	obsactie gamesex is	seus popureune 4 mo	3-sae vapilles
9. Resolver	испучает ссыми с фотива (деспрумо	na paggemeeneme	pecypen, us. ounecu	cegyet.
10. Umryus	um zepycice anna pari	un uenteux noque	ins nutries was a few	e sussal en s
11. Ramos	To yource not acres	u l'orepego raroba.	une nyrever non poberna	a pemajor pegular
12. 12038p.	1313 6 vorus leozopa	ra 43 micremnoro	buzola 6 pogurence	uou notone
процесс	e u 0-b upage	ece-notocure.		

Рисунок 6.1 – Конспект по системному вызову fork()

На рис. 6.2 приведен конспект по системному вызову ехес().

```
Mysmol
   exec
   1. Passayaet myte a new oponing a organization est goetyn a weelig
   2. Probapors, aunes un Consubanouguis apreject nonsenereme na bours-un apartire,
   3. Marait zaracobon u godiepert, 200 on genemberreceno una membero.
4. Com que apañ sa yeta nobuena Suron SUSD una SGID, 10 reposentiebase
   ugus aperua soper USD a GID bezubaroryero apoyerca uzur A na UID a
   GID, coop b. buc genting governo
  5. Comput aprimente, a son de get apigabaciene le exec, a sanse
   перешенине срезог в пробраненье изрог, поше чего чекущее папозовобенения
  upo a pomembo rotoleo in juerto semuio
 6. Buqueles upor parembo chommuna que odiac sen gamens a crena
 3. Borcho Josquem crapor agreence up bo i chezanuse i una opocupamento chommuna Eau oc apocações sun cozona que nomocare Norte apour hoguital les aport craporo agrecuoro apocarpanente possivens onocus apocación.
s. Bugune un napro spancine yen agreed qui notado serina, gannon u escue
9. Jesa naticularin nobose agrecuse upos pomembre. Ecun Dears venere autubas.
 apour parentes gourno mun que un pobertice es banacareccios goartes
 Mayeur le avoience UNIX odurno parduore na expansión, en oquaract, no nasque expansión contribuence le naceurs toresto no nepe mode re.
со, конција признентог и перешениме среда обратио в повоит стен прешохони
11. Dopacabaem bee odpadotreum curnamol l'genembano, aup me no yeromanne
тан пак пиструпции эбрабоченов синанов не сущем вучет в новой програми
 виспания, почорие быт произпорирование ими забионирование при перез
buzdour exec, octoursul to sex se coctonennes
17. Uniqually zupyet anna partient nonvencin. Pyen river Societicemen bo
 peningob especularina 60, a jul joiselle homang nongreen quarenne
Torum broge reorganiche
```

Рисунок 6.2 – Конспект по системному вызову ехес()