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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 2

Название: Процессы. Системные вызовы.

Дисциплина: Операционные системы

Студент	ИУ7-52Б		Е.В. Брянская
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Н.Ю. Рязанова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Задание 1.

Написать программу, запускающую не менее двух новых процессов системным вызовом fork().

В предке вывести собственный идентификатор (функция getpid()), идентификатор группы (функция getpgrp()) и идентификаторы потомков.

В процессе-потомке вывести собственный идентификатор, идентификатор предка (функция getppid()) и идентификатор группы.

Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

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

void child_action()
{
    printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
    sleep(1);
    printf("\nChild:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
}

int main(int argc, char *argv[])
{
    pid_t childpid_1 = fork(), childpid_2;

    if (childpid_1 == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid_1 == 0)
    {
        child_action();
        return 0;
    }

    childpid_2 = fork();

    if (childpid_2 == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid_2 == 0)
    {
        child_action();
        return 0;
    }

    printf("Parent: id = %d group_id  = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid_1, childpid_2);

    return 0;
}
```

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_1.o
Parent: id = 6381       group_id  = 6381        children = 6382 6383
Child:  id = 6382       parent_id = 6381        group_id = 6381
Child:  id = 6383       parent_id = 6381        group_id = 6381
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$
Child:  id = 6382       parent_id = 1523        group_id = 6381

Child:  id = 6383       parent_id = 1523        group_id = 6381
```

Задание 2.

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов wait(). Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

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


void child_action()
{
    printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
}

int main(int argc, char *argv[])
{
    pid_t childpid_1 = fork(), childpid_2;
    int status;

    if (childpid_1 == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid_1 == 0)
    {
        child_action();
        return 0;
    }

    childpid_2 = fork();

    if (childpid_2 == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid_2 == 0)
    {
        child_action();
        return 0;
    }

    printf("Parent: id = %d group_id  = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid_1, childpid_2);

```

```c
    for (int i = 0; i < 2; i++)
    {
        printf("\n--- Parent is waiting ---");

        pid_t childpid = wait(&status);
        if (childpid == -1)
        {
            if (errno == ECHILD)
                printf("Process does not have any unwaited for children\n");
            else if (errno == EINTR)
                printf("Call interrupted by signal\n");
            else if (errno == EINVAL)
                printf("Wrong argument\n");
            exit(1);
        }

        printf("\nChild finished: pid = %d\n", childpid);

        if (WIFEXITED(status))
            printf("Child exited normally with code %d\n", WEXITSTATUS(status));
        else printf("Child terminated abnormally\n");

        if (WIFSIGNALED(status))
            printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

        if (WIFSTOPPED(status))
            printf("Child stopped, signal # %d\n", WSTOPSIG(status));
    }

    return 0;
}
```

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_2.o
Parent: id = 6582       group_id  = 6582        children = 6583 6584

Child:  id = 6583       parent_id = 6582        group_id = 6582
Child:  id = 6584       parent_id = 6582        group_id = 6582
--- Parent is waiting ---
Child finished: pid = 6583
Child exited with code 0

--- Parent is waiting ---
Child finished: pid = 6584
Child exited with code 0
```

Задание 3.

Написать программу, в которой процесс-потомок вызывает системный вызов exec(), а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

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


int main(int argc, char *argv[])
{
    pid_t childpid = fork();
    int status;
    char temp[10];

    if (childpid == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid == 0)
    {
        printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());

        sprintf(temp, "%d", getpid());
        status = execl("add_program_1.o", "add_program_1.o", temp, NULL);
        if (status == -1)
        {
            printf("Execl error");
            exit(1);
        }

        return 0;
    }

    printf("Parent: id = %d group_id  = %d\n", getpid(), getpgrp());

    childpid = fork();

    if (childpid == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid == 0)
    {
        printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());

        status = execl("add_program_2.o", "", NULL);
        if (status == -1)
        {
            printf("Execl error");
            exit(1);
        }

        return 0;
    }

    for (int i = 0; i < 2; i++)
    {
        printf("\n--- Parent is waiting ---");

```

```
69          childpid = wait(&status);
70          if (childpid == -1)
71          {
72              if (errno == ECHILD)
73                  printf("Process does not have any unwaited for children\n");
74              else if (errno == EINTR)
75                  printf("Call interrupted by signal\n");
76              else if (errno == EINVAL)
77                  printf("Wrong argument\n");
78              exit(1);
79          }
80
81          printf("\nChild finished: pid = %d\n", childpid);
82
83          if (WIFEXITED(status))
84              printf("Child exited normally with code %d\n", WEXITSTATUS(status));
85          else printf("Child terminated abnormally\n");
86
87          if (WIFSIGNALED(status))
88              printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));
89
90          if (WIFSTOPPED(status))
91              printf("Child stopped, signal # %d\n", WSTOPSIG(status));
92      }
93
94      return 0;
95  }
```

Вызываемые программы:

- add_program_1.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(int argc, char *argv[])
5   {
6       if (argc != 2)
7       {
8           printf("\n\n--> Error: wrong number of arguments!\n");
9           exit(1);
10      }
11
12      printf("\n\n--> Hello, my id is %s\n", argv[1]);
13
14      return 0;
15  }
```

- add_program_2.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(int argc, char *argv[])
5   {
6       printf("\n--> ");
7       for (int i = 0; i < 10; i++)
8           printf("%d ", i);
9       printf("\n\n");
10
11      return 0;
12  }
```

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_3.o
Parent: id = 3692       group_id  = 3692
Child:  id = 3693       parent_id = 3692        group_id = 3692

Child:  id = 3694       parent_id = 3692        group_id = 3692


--> Hello, my id is 3693

--> 0 1 2 3 4 5 6 7 8 9

--- Parent is waiting ---
Child finished: pid = 3693
Child exited with code 0

--- Parent is waiting ---
Child finished: pid = 3694
Child exited with code 0
```

Задание 4.

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

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


int main(int argc, char *argv[])
{
    int fd[2];
    int status;
    char get_msg[25], send_msg[25];

    if (pipe(fd) == -1)
    {
        perror("Can't pipe");
        exit(1);
    }

    pid_t childpid = fork();

    if (childpid == -1)
    {
        perror("Can't fork");
        exit(1);
    }

    if (childpid == 0)
    {
        if (close(fd[0]) == -1)
        {
            printf("Close error");
            exit(1);
        }

        printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());

```

```
43          sprintf(send_msg, "Hello, it's child %d", getpid());
44          status = write(fd[1], send_msg, sizeof(send_msg));
45          if (status == -1)
46          {
47              printf("Write error");
48              exit(1);
49          }
50
51          if (close(fd[1]) == -1)
52          {
53              printf("Close error");
54              exit(1);
55          }
56
57          sleep(1);
58          printf("\n- Child %d sent: %s\n", getpid(), send_msg);
59
60          return 0;
61      }
62
63      printf("Parent: id = %d group_id  = %d\n", getpid(), getpgrp());
64
65      childpid = fork();
66
67      if (childpid == -1)
68      {
69          perror("Can't fork");
70          exit(1);
71      }
72
73      if (childpid == 0)
74      {
75          if (close(fd[0]) == -1)
76          {
77              printf("Close error");
78              exit(1);
79          }
80
81          printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
82
83           sprintf(send_msg, "Hello, it's child %d", getpid());
84           status = write(fd[1], send_msg, sizeof(send_msg));
85           if (status == -1)
86           {
87               printf("Write error");
88               exit(1);
89           }
90
91           if (close(fd[1]) == -1)
92           {
93               printf("Close error");
94               exit(1);
95           }
96
97           sleep(1);
98           printf("\n- Child %d sent: %s\n", getpid(), send_msg);
99
100          return 0;
101      }
102
103      if (close(fd[1]) == -1)
104      {
105          printf("Close error");
106          exit(1);
107      }
108
```

```
    for (int i = 0; i < 2; i++)
    {
        childpid = wait(&status);
        if (childpid == -1)
        {
            if (errno == ECHILD)
                printf("Process does not have any unwaited for children\n");
            else if (errno == EINTR)
                printf("Call interrupted by signal\n");
            else if (errno == EINVAL)
                printf("Wrong argument\n");
            exit(1);
        }

        printf("\nChild finished: pid = %d\n", childpid);

        if (WIFEXITED(status))
            printf("Child exited normally with code %d\n", WEXITSTATUS(status));
        else printf("Child terminated abnormally\n");

        if (WIFSIGNALED(status))
            printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

        if (WIFSTOPPED(status))
            printf("Child stopped, signal # %d\n", WSTOPSIG(status));
    }

    status = read(fd[0], get_msg, sizeof(get_msg));
    if (status == -1)
    {
        printf("Read 1 error");
        exit(1);
    }

    printf("\n--> Parent read: %s\n", get_msg);

    status = read(fd[0], get_msg, sizeof(get_msg));
    if (status == -1)
    {
        printf("Read 2 error");
        exit(1);
    }

    printf("\n--> Parent read: %s\n", get_msg);

    if (close(fd[0]) == -1)
    {
        printf("Close error");
        exit(1);
    }

    return 0;
}
```

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_4.o
Parent: id = 3881       group_id  = 3881
Child:  id = 3882       parent_id = 3881        group_id = 3881
Child:  id = 3883       parent_id = 3881        group_id = 3881

- Child 3882 sent: Hello, it's child 3882

Child finished: pid = 3882
Child exited with code 0

- Child 3883 sent: Hello, it's child 3883

Child finished: pid = 3883
Child exited with code 0

--> Parent read: Hello, it's child 3882

--> Parent read: Hello, it's child 3883
```

Задание 5.

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

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


int send_signal_1 = 0, send_signal_2 = 0;


void action_1(int sig_numb)                // Ctrl-z
{
    send_signal_1 = 1;
}

void action_2(int sig_numb)                // Ctrl-'\'
{
    send_signal_2 = 1;
}

int main(int argc, char *argv[])
{
    int fd[2];
    int status;
    char get_msg[25], send_msg[25];
    pid_t childpid_1, childpid_2;

    if (signal(SIGTSTP, action_1) == SIG_ERR || signal(SIGQUIT, action_2) == SIG_ERR)    // Ctrl-z и Ctrl-'\'
    {
        printf("Signal error");
        exit(1);
    }

    if (pipe(fd) == -1)
    {
        perror("Can't pipe");
        exit(1);
    }

```

```
46      childpid_1 = fork();
47
48      if (childpid_1 == -1)
49      {
50          perror("Can't fork");
51          exit(1);
52      }
53
54      if (childpid_1 == 0)
55      {
56          if (close(fd[0]) == -1)
57          {
58              printf("Close error");
59              exit(1);
60          }
61
62          printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
63
64          if (signal(SIGQUIT, SIG_IGN) == SIG_ERR)        // Ctrl-'\'
65          {
66              printf("Signal error");
67              exit(1);
68          }
69
70          sleep(5);
71
72          if (send_signal_1)
73          {
74              sprintf(send_msg, "Hello, it's child %d", getpid());
75              status = write(fd[1], send_msg, sizeof(send_msg));
76              if (status == -1)
77              {
78                  printf("Write error");
79                  exit(1);
80              }
81
82              printf("\n- Child %d sent: %s\n", getpid(), send_msg);
83          }
84
85          if (close(fd[1]) == -1)
86          {
87              printf("Close error");
88              exit(1);
89          }
90
91          return 0;
92      }
93
94      printf("Parent: id = %d group_id  = %d\n", getpid(), getpgrp());
95
96      childpid_2 = fork();
97
98      if (childpid_2 == -1)
99      {
100         perror("Can't fork");
101         exit(1);
102     }
103
104     if (childpid_2 == 0)
105     {
106         if (close(fd[0]) == -1)
107         {
108             printf("Close error");
109             exit(1);
110         }
111
112         printf("Child:  id = %d \tparent_id = %d \tgroup_id = %d\n", getpid(), getppid(), getpgrp());
113
114         if (signal(SIGTSTP, SIG_IGN) == SIG_ERR)        // Ctrl-z
115         {
116             printf("Signal error");
117             exit(1);
118         }
119
120         sleep(5);
121
```

```
122         if (send_signal_2)
123         {
124             sprintf(send_msg, "Hello, it's child %d", getpid());
125             status = write(fd[1], send_msg, sizeof(send_msg));
126             if (status == -1)
127             {
128                 printf("Write error");
129                 exit(1);
130             }
131 
132             printf("\n- Child %d sent: %s\n", getpid(), send_msg);
133         }
134 
135         if (close(fd[1]) == -1)
136         {
137             printf("Close error");
138             exit(1);
139         }
140 
141         return 0;
142     }
143 
144     if (close(fd[1]) == -1)
145     {
146         printf("Close error");
147         exit(1);
148     }
149 
150     for (int i = 0; i < 2; i++)
151     {
152         pid_t childpid = wait(&status);
153         if (childpid == -1)
154         {
155             if (errno == ECHILD)
156                 printf("Process does not have any unwaited for children\n");
157             else if (errno == EINTR)
158                 printf("Call interrupted by signal\n");
159             else if (errno == EINVAL)
160                 printf("Wrong argument\n");
161             exit(1);
162         }
163 
164         printf("\nChild finished: pid = %d\n", childpid);
165 
166         if (WIFEXITED(status))
167             printf("Child exited normally with code %d\n", WEXITSTATUS(status));
168         else printf("Child terminated abnormally\n");
169 
170         if (WIFSIGNALED(status))
171             printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));
172 
173         if (WIFSTOPPED(status))
174             printf("Child stopped, signal # %d\n", WSTOPSIG(status));
175     }
176 
177     if (send_signal_1)
178     {
179         status = read(fd[0], get_msg, sizeof(get_msg));
180         if (status == -1)
181         {
182             printf("Read 1 error");
183             exit(1);
184         }
185 
186         printf("\n--> Parent read: %s\n", get_msg);
187     }
188 
```

```
189     if (send_signal_2)
190     {
191         status = read(fd[0], get_msg, sizeof(get_msg));
192         if (status == -1)
193         {
194             printf("Read 2 error");
195             exit(1);
196         }
197 
198         printf("\n--> Parent read: %s\n", get_msg);
199     }
200 
201     if (close(fd[0]) == -1)
202     {
203         printf("Close error");
204         exit(1);
205     }
206 
207     return 0;
208 }
```

Набрана комбинация Ctrl-z:

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_6.o
Parent: id = 7736       group_id  = 7736
Child:  id = 7737       parent_id = 7736        group_id = 7736
Child:  id = 7738       parent_id = 7736        group_id = 7736
^Z
- Child 7737 sent: Hello, it's child 7737

Child finished: pid = 7737
Child exited normally with code 0

Child finished: pid = 7738
Child exited normally with code 0

--> Parent read: Hello, it's child 7737
```

Набрана комбинация Ctrl-\:

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/lab_02$ ./task_6.o
Parent: id = 7765       group_id  = 7765
Child:  id = 7766       parent_id = 7765        group_id = 7765
Child:  id = 7767       parent_id = 7765        group_id = 7765
^\
- Child 7767 sent: Hello, it's child 7767

Child finished: pid = 7767
Child exited normally with code 0

Child finished: pid = 7766
Child exited normally with code 0

--> Parent read: Hello, it's child 7767
```

Набраны обе комбинации: