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»

Отчёт

по лабораторной работе № 4

Название: Пять системных вызовов ОС UNIX/LINUX

Дисциплина: Операционные системы

Студент ИУ7-55Б _____ Д.О. Склифасовский
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Н.Ю. Рязанова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Задание 1

Написать программу, запускающую не мене 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.

Программа:

Листинг 1 – Задание 1

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

void print_child(int child_num, char *descr)
{
    printf("child: number=%d pid=%d parent=%d group=%d %s\n",
        child_num, getpid(), getppid(), getpgrp(), descr);
}

pid_t fork_child(int child_num)
{
    pid_t child = fork();
    if (child == -1)
    {
        perror("fork");
        exit(1);
    }
    else if (child == 0)
    {
```

```
20          print_child(child_num, "before sleep");
21          sleep(2);
22          print_child(child_num, "after sleep");
23          exit(0);
24      }
25      return child;
26  }
27
28  int main()
29  {
30      pid_t child_1 = fork_child(1);
31      pid_t child_2 = fork_child(2);
32
33      printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
            getpid(), getpgrp(), child_1, child_2);
34
35      return 0;
36  }
```

Результат работы программы:

```
parent: pid=6427, group=6427, child_1=6428, child_2=6429
child: number=1 pid=6428 parent=6427 group=6427 before sleep
child: number=2 pid=6429 parent=6427 group=6427 before sleep
mrskl1f@mrskl1f-ThinkPad-E595:~/Рабочий стол/BMSTU/OS/Lab2$ child: number=1 pid=
6428 parent=1571 group=6427 after sleep
child: number=2 pid=6429 parent=1571 group=6427 after sleep
```

Рисунок 1 – Результат работы программы

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.

Код программы:

Листинг 2 – Задание 2

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

void print_child(int child_num)
{
    printf("child: number=%d pid=%-5d parent=%-5d group=%-5d\n",
        child_num, getpid(), getppid(), getpgrp());
}

pid_t fork_child(int child_num)
{
    pid_t child = fork();
    if (child == -1)
    {
        perror("fork");
        exit(1);
    }
    else if (child == 0)
    {
        print_child(child_num);
        exit(0);
    }
    return child;
}
```

```
27
28 void wait_for_childs()
29 {
30     int stat_val;
31     pid_t child = wait(&stat_val);
32     printf("Child has finished: PID=%d\n", child);
33     if (WIFEXITED(stat_val))
34         printf("Child=%d completed normally with code=%d.\n",
              child, WEXITSTATUS(stat_val));
35     else if (WIFSIGNALED(stat_val))
36         printf("Child=%d ended with a non-intercepted signal with
              code=%d.\n", child, WTERMSIG(stat_val));
37     else if (WIFSTOPPED(stat_val))
38         printf("Child=%d stopped with %d code.\n", child,
              WSTOPSIG(stat_val));
39 }
40
41 int main()
42 {
43     pid_t child_1 = fork_child(1);
44     pid_t child_2 = fork_child(2);
45
46     printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
          getpid(), getpgrp(), child_1, child_2);
47
48     wait_for_childs();
49     wait_for_childs();
50
51     return 0;
52 }
```

Результат работы программы:

```
parent: pid=6536, group=6536, child_1=6537, child_2=6538
child: number=1 pid=6537  parent=6536  group=6536
child: number=2 pid=6538  parent=6536  group=6536
Child has finished: PID=6537
Child=6537 completed normally with code=0.
Child has finished: PID=6538
Child=6538 completed normally with code=0.
```

Рисунок 2 – Результат работы программы

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.

Код программы:

Листинг 3 – Задание 3

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

void print_child(int child_num)
{
    printf("child: number=%d pid=%-5d parent=%-5d group=%-5d\n",
        child_num, getpid(), getppid(), getpgrp());
}

pid_t fork_child(int child_num, char *path, char *arg0)
{
    pid_t child = fork();
    if (child == -1)
    {
        perror("fork");
        exit(1);
    }
    else if (child == 0)
    {
        print_child(child_num);
        if (execl(path, arg0, NULL) == -1)
        {
            perror("exec");
            exit(1);
```

```
27          }
28
29      }
30      return child;
31  }
32
33  void wait_for_childs()
34  {
35      int stat_val;
36      pid_t child = wait(&stat_val);
37      printf("Child has finished: PID=%d\n", child);
38      if (WIFEXITED(stat_val))
39          printf("Child=%d completed normally with code=%d.\n",
              child, WEXITSTATUS(stat_val));
40      else if (WIFSIGNALED(stat_val))
41          printf("Child=%d ended with a non-intercepted signal with
                code=%d.\n", child, WTERMSIG(stat_val));
42      else if (WIFSTOPPED(stat_val))
43          printf("Child=%d stopped with %d code.\n", child,
              WSTOPSIG(stat_val));
44  }
45
46  int main()
47  {
48      pid_t child_1 = fork_child(1, "/bin/ls", "ls");
49      pid_t child_2 = fork_child(2, "/bin/ps", "ps");
50
51      printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
          getpid(), getpgrp(), child_1, child_2);
52
53      wait_for_childs();
54      wait_for_childs();
55
56      return 0;
57  }
```

Результат работы программы:

```
child: number=1 pid=6902  parent=6901  group=6901
parent: pid=6901, group=6901, child_1=6902, child_2=6903
child: number=2 pid=6903  parent=6901  group=6901
prog1.c  prog2.c  prog3.c  prog4.c  prog5.c  prog.out  Report
Child has finished: PID=6902
Child=6902 completed normally with code=0.
  PID TTY          TIME CMD
 6402 pts/1    00:00:00 bash
 6901 pts/1    00:00:00 prog.out
 6903 pts/1    00:00:00 ps
Child has finished: PID=6903
Child=6903 completed normally with code=0.
```

Рисунок 3 – Результат работы программы

Задание 4

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

Код программы:

Листинг 4 – Задание 4

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

void reverse(char s[])
{
    int i, j;
    char c;

    for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void convert_to_str(int n, char s[])
{
    int i = 0;
    do
    {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
    s[i] = '\0';
```

```
29      reverse(s);
30  }
31
32
33  pid_t fork_child(int child_num, int *fd)
34  {
35      pid_t child = fork();
36      char pid[10];
37      if (child == -1)
38      {
39          perror("fork");
40          exit(1);
41      }
42      else if (child == 0)
43      {
44          convert_to_str(getpid(), pid);
45          close(fd[0]);
46          write(fd[1], pid, sizeof(pid));
47          exit(0);
48      }
49      return child;
50  }
51
52  void wait_for_childs(int *fd)
53  {
54      int stat_val;
55      char pid[10];
56      read(fd[0], pid, 10);
57      pid_t child = wait(&stat_val);
58      printf("Child %d wrote %s\n", child, pid);
59      if (WIFEXITED(stat_val))
60          printf("Child=%d completed normally with code=%d.\n",
                child, WEXITSTATUS(stat_val));
61      else if (WIFSIGNALED(stat_val))
62          printf("Child=%d ended with a non-intercepted signal with
```

```
            code=%d.\n", child, WTERMSIG(stat_val));
63      else if (WIFSTOPPED(stat_val))
64          printf("Child=%d stopped with %d code.\n", child,
              WSTOPSIG(stat_val));
65  }
66
67  int main()
68  {
69      int fd[2];
70      if (pipe(fd) == -1)
71      {
72          perror("pipe");
73          exit(1);
74      }
75
76      pid_t child_1 = fork_child(1, fd);
77      pid_t child_2 = fork_child(2, fd);
78
79      printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
          getpid(), getpgrp(), child_1, child_2);
80      close(fd[1]);
81      wait_for_childs(fd);
82      wait_for_childs(fd);
83
84      return 0;
85  }
```

Результат работы программы:

```
parent: pid=7080, group=7080, child_1=7081, child_2=7082
Child 7081 wrote 7081
Child=7081 completed normally with code=0.
Child 7082 wrote 7082
Child=7082 completed normally with code=0.
```

Рисунок 4 – Результат работы программы

Задание 5

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

Код программы:

Листинг 5 – Задание 5

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

int flag = 0;
int fd[2];

void reverse(char s[])
{
    int i, j;
    char c;

    for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void convert_to_str(int n, char s[])
{
    int i = 0;
    do
```

```
28      {
29          s[i++] = n % 10 + '0';
30      } while ((n /= 10) > 0);
31      s[i] = '\0';
32      reverse(s);
33  }
34
35  void catch_signal(int sig_numb)
36  {
37      printf("\nSignal Cntrl + C\n");
38      if (flag)
39      flag = 0;
40      else
41      flag = 1;
42  }
43
44  void wait_signal(char *operation)
45  {
46      printf("Press Ctrl + C to %s.\n", operation);
47      signal(SIGINT, catch_signal);
48      sleep(5);
49  }
50
51  pid_t fork_child(int child_num)
52  {
53      pid_t child = fork();
54      char pid[10];
55      if (child == -1)
56      {
57          perror("fork");
58          exit(1);
59      }
60      else if (child == 0)
61      {
62          convert_to_str(getpid(), pid);
```

```
63          close(fd[0]);
64          write(fd[1], pid, sizeof(pid));
65          exit(0);
66      }
67      return child;
68  }
69
70  void wait_for_childs(pid_t child, int stat_val)
71  {
72      char pid[10];
73      read(fd[0], pid, 10);
74      printf("Child %d wrote %s\n", child, pid);
75      if (WIFEXITED(stat_val))
76      printf("Child=%d completed normally with code=%d.\n", child,
            WEXITSTATUS(stat_val));
77      else if (WIFSIGNALED(stat_val))
78      printf("Child=%d ended with a non-intercepted signal with
            code=%d.\n", child, WTERMSIG(stat_val));
79      else if (WIFSTOPPED(stat_val))
80      printf("Child=%d stopped with %d code.\n", child, WSTOPSIG(
            stat_val));
81  }
82
83  void parent_read(pid_t child_1, pid_t child_2)
84  {
85      if (child_1 != 0 && child_2 != 0)
86      {
87          int stat_val_1;
88          int stat_val_2;
89          pid_t child_1 = wait(&stat_val_1);
90          pid_t child_2 = wait(&stat_val_2);
91
92          wait_signal("read");
93          if (flag)
94          exit(0);
```

```
95
96          close(fd[1]);
97          wait_for_childs(child_1, stat_val_1);
98          wait_for_childs(child_2, stat_val_2);
99      }
100 }
101
102 int main()
103 {
104     if (pipe(fd) == -1)
105     {
106         perror("pipe");
107         exit(1);
108     }
109
110     wait_signal("write");
111
112     if (!flag)
113     exit(0);
114
115     pid_t child_1 = fork_child(1);
116     pid_t child_2 = fork_child(2);
117     printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
            getpid(), getpgrp(), child_1, child_2);
118     parent_read(child_1, child_2);
119     return 0;
120 }
```

Результат работы программы:

```
Press Ctrl + C to write.
^C
Signal Cntrl + C
parent: pid=7223, group=7223, child_1=7224, child_2=7225
Press Ctrl + C to read.
^C
Signal Cntrl + C
Child 7224 wrote 7224
Child=7224 completed normally with code=0.
Child 7225 wrote 7225
Child=7225 completed normally with code=0.
```

Рисунок 5 – Результат работы программы