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 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.

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>

pid_t fork_child(int child_num, int *fd)
{
    pid_t child = fork();
    if (child == -1)
    {
        perror("fork");
        exit(1);
    }
    else if (child == 0)
    {
        int child_pid = getpid();
        void *pid = &child_pid;
        close(fd[0]);
        write(fd[1], pid, sizeof(void *));
        exit(0);
    }
    return child;
}

void wait_for_childs(int *fd)
{
    close(fd[1]);
```

```
29
30      int stat_val;
31      void *pid = (void *)malloc(sizeof(void *));
32      read(fd[0], pid, sizeof(pid));
33
34      pid_t child = wait(&stat_val);
35      printf("Child %d wrote %d\n", child, *(int *)(pid));
36      if (WIFEXITED(stat_val))
37      printf("Child=%d completed normally with code=%d.\n", child,
           WEXITSTATUS(stat_val));
38      else if (WIFSIGNALED(stat_val))
39      printf("Child=%d ended with a non-intercepted signal with
           code=%d.\n", child, WTERMSIG(stat_val));
40      else if (WIFSTOPPED(stat_val))
41      printf("Child=%d stopped with %d code.\n", child, WSTOPSIG(
           stat_val));
42      free(pid);
43  }
44
45  int main()
46  {
47      int fd[2];
48      if (pipe(fd) == -1)
49      {
50          perror("pipe");
51          exit(1);
52      }
53
54      pid_t child_1 = fork_child(1, fd);
55      pid_t child_2 = fork_child(2, fd);
56
57      printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
           getpid(), getpgrp(), child_1, child_2);
58      wait_for_childs(fd);
59      wait_for_childs(fd);
```

```
60
61      return 0;
62  }
```

Результат работы программы:

```
parent: pid=128, group=128, child_1=129, child_2=130
Child 129 wrote 129
Child=129 completed normally with code=0.
Child 130 wrote 130
Child=130 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>

int flag = 0;

void catch_signal(int sig_numb)
{
    printf("\nSignal Ctrl + C\n");
    flag = 1;
}

void wait_signal()
{
    printf("Press Ctrl + C to write.\n");
    signal(SIGINT, catch_signal);
    sleep(3);
}

pid_t fork_child(int child_num, int *fd)
{
    pid_t child = fork();
    if (child == -1)
    {
        perror("fork");
```

```
28          exit(1);
29      }
30      else if (child == 0)
31      {
32          if (flag)
33          {
34              int child_pid = getpid();
35              void *pid = &child_pid;
36              close(fd[0]);
37              write(fd[1], pid, sizeof(pid));
38          }
39          exit(0);
40      }
41      return child;
42  }
43
44  void wait_for_childs(int *fd)
45  {
46      int stat_val;
47      void *pid = (void *)malloc(sizeof(void *));
48
49      pid_t child = wait(&stat_val);
50
51      if (flag)
52      {
53          close(fd[1]);
54          read(fd[0], pid, sizeof(pid));
55          printf("Child %d wrote %d\n", child, *(int *)(pid));
56      }
57      else
58      printf("Child %d wrote nothing\n", child);
59
60      if (WIFEXITED(stat_val))
61      printf("Child=%d completed normally with code=%d.\n", child,
          WEXITSTATUS(stat_val));
```

```
62      else if (WIFSIGNALED(stat_val))
63      printf("Child=%d ended with a non-intercepted signal with
            code=%d.\n", child, WTERMSIG(stat_val));
64      else if (WIFSTOPPED(stat_val))
65      printf("Child=%d stopped with %d code.\n", child, WSTOPSIG(
            stat_val));
66      free(pid);
67  }
68
69  int main()
70  {
71      int fd[2];
72      if (pipe(fd) == -1)
73      {
74          perror("pipe");
75          exit(1);
76      }
77
78      wait_signal();
79
80      pid_t child_1 = fork_child(1, fd);
81      pid_t child_2 = fork_child(2, fd);
82
83      printf("parent: pid=%d, group=%d, child_1=%d, child_2=%d\n",
            getpid(), getpgrp(), child_1, child_2);
84
85      wait_for_childs(fd);
86      wait_for_childs(fd);
87
88      return 0;
89  }
```

Результат работы программы 1:

```
Press Ctrl + C to write.
^C
Signal Ctrl + C
parent: pid=136, group=136, child_1=137, child_2=138
Child 137 wrote 137
Child=137 completed normally with code=0.
Child 138 wrote 138
Child=138 completed normally with code=0.
```

Рисунок 5 – Результат работы программы 1

Результат работы программы 2:

```
Press Ctrl + C to write.
parent: pid=139, group=139, child_1=140, child_2=141
Child 140 wrote nothing
Child=140 completed normally with code=0.
Child 141 wrote nothing
Child=141 completed normally with code=0.
```

Рисунок 6 – Результат работы программы 2