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 №4 по дисциплине "Операционные системы"

Тема Процессы. Системные вызовы fork() и exec()

Студент Варламова Е. А.

Группа ИУ7-51Б

Оценка (баллы) _____

Преподаватели Рязанова Н. Ю.

Москва — 2021 г.

Задание №1

Процессы-сироты. В программе создаются не менее двух потомков системным вызовом fork(). В потомках вызывается sleep(), чтобы предок гарантированно завершился раньше своих потомков. В предке вывести собственный идентификатор, идентификатор группы и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника. 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.

В программе добавлен sleep в предке перед его завершением для того, чтобы предок не завершился до того, как будет выведена информация о процессах-потомках до блокировки. При этом время блокировки предка меньше, чем время блокировки потомков, поэтому предок гарантированно завершится раньше своих потомков.

Текст программы приведён на листинге 1.

Листинг 1: Процессы-сироты

```
#include <stdio.h>
#include <unistd.h>
#define ERROR_FORK 1
#define OK 0
int main() {
    int childpids[2];
    for (int i = 0; i < 2; i++) {
        int pid = fork();

        if (pid == -1) {
            return ERROR_FORK;
        }

        if (pid == 0) {
            printf("\nCHILD    %d LOG BEFORE BLOCK pid: %d, ppid: %d, grp: %
               d\n", i + 1, getpid(), getppid(), getpgrp());
            sleep(2);
            printf("\nCHILD    %d LOG AFTER BLOCK pid: %d, ppid: %d, grp: %d
               \n", i + 1, getpid(), getppid(), getpgrp());
            return OK;
        }

        childpids[i] = pid;
    }
    printf("PARENT pid: %d grp: %d, child's pids: %d, %d\n", getpid(),
       getpgrp(), childpids[0], childpids[1]);
    sleep(1);
    return OK;
}
```

На риснуке 1 приведён результат работы программы. Видно, что до завершения процесса-предка ppid у потомков был равен идентификатору предка. Затем, когда процесс-предок завершился, потомки были "усыновлены"процессом с идентификатором 1.

```
MacBook-Pro-Ekaterina:lab_04_01 kate$ ./app
PARENT pid: 43022 grp: 43022, child's pids: 43023, 43024

CHILD №1 LOG BEFORE BLOCK pid: 43023, ppid: 43022, grp: 43022

CHILD №2 LOG BEFORE BLOCK pid: 43024, ppid: 43022, grp: 43022
MacBook-Pro-Ekaterina:lab_04_01 kate$
CHILD №1 LOG AFTER BLOCK pid: 43023, ppid: 1, grp: 43022

CHILD №2 LOG AFTER BLOCK pid: 43024, ppid: 1, grp: 43022
```

Рис. 1: Демонстрация работы программы (задание №1).

Задание №2

Предок ждет завершения своих потомком, используя системный вызов wait(). Вывод соответствующих сообщений на экран. В программе необходимо, чтобы предок выполнял анализ кодов завершения потомков. Текст программы приведён на листинге 2.

Листинг 2: Системный вызов wait()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define ERROR_FORK 1
#define OK 0
int main() {
    int childpids[2];
    for (int i = 0; i < 2; i++) {
        int pid = fork();
        if (pid == -1) {
            return ERROR_FORK;
        }
        if (pid == 0){
            sleep(2);
            printf("CHILD    %d LOG pid: %d, ppid: %d, grp: %d\n", i + 1,
                getpid(), getppid(), getpgrp());
            return OK;
        }
        childpids[i] = pid;
    }
    printf("PARENT LOG pid: %d grp: %d, child's pids: %d, %d\n", getpid(),
       getpgrp(), childpids[0], childpids[1]);
    for (int i = 0; i < 2; i++)
    {
        int status;
        pid_t childpid = wait(&status);
        if (WIFEXITED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has finished with code
                : %d\n", i + 1, childpid, WEXITSTATUS(status));
        }
        else if (WIFSIGNALED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has finished because
                of signal: %d\n", i + 1, childpid, WTERMSIG(status));
        }
        else if (WIFSTOPPED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has been stopped
                because of signal: %d\n", i + 1, childpid, WSTOPSIG(status));
        }
    }
    return OK;
}
```

Результат работы программы приведён на рисунке 2. Видно, что в отличие от программы из первого задания процесс-предок дождался завершения дечерних процессов, о чём свидетельствуют коды завершения дочерних процессов, перехваченные процессом-предком.

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_02 (master)> ./app
PARENT LOG pid: 44107 grp: 44107, child's pids: 44108, 44109
CHILD №1 LOG pid: 44108, ppid: 44107, grp: 44107
CHILD №2 LOG pid: 44109, ppid: 44107, grp: 44107
PARENT LOG child has finished: PID = 44109, status: 0
PARENT LOG child has finished: PID = 44108, status: 0
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_02 (master)>
```

Рис. 2: Демонстрация работы программы (задание №2).

Задание №3

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. Текст программы приведён на листинге 3.

Листинг 3: Системный вызов exec()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define ERROR_FORK 1
#define ERROR_EXEC 2
#define OK 0
int main() {
    int childpids[2];
    for (int i = 0; i < 2; i++) {
        int pid = fork();
        if (pid == -1) {
            return ERROR_FORK;
        }
        if (pid == 0) {
            printf("CHILD    %d LOG pid: %d, ppid: %d, grp: %d\n", i + 1,
                getpid(), getppid(), getpgrp());
            if (i == 0) execlp("ps", "-al", NULL);
            else execlp("ls", "-a", NULL);
            return ERROR_EXEC;
        }
        childpids[i] = pid;
    }
    printf("PARENT LOG pid: %d grp: %d, child's pids: %d, %d\n", getpid(),
       getpgrp(), childpids[0], childpids[1]);
    for (int i = 0; i < 2; i++) {
        int status;
        pid_t childpid = wait(&status);
        if (WIFEXITED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has finished with code
                : %d\n", i + 1, childpid, WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has finished because
                of signal: %d\n", i + 1, childpid, WTERMSIG(status));
        } else if (WIFSTOPPED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has been stopped
                because of signal: %d\n", i + 1, childpid, WSTOPSIG(status));
        }
    }
    return OK;
}
```

Результат работы программы приведён на рисунке 3. Видно, что в отличие от программы из второго задания дочерние процессы переходят на выполнение других программ, которые передаются системному вызову exec() в качестве параметра. В данном случае выполняются программы "ps - al"и "ls -a". При этом код, который следует за вызовом exec() будет выполнен лишь в том случае, если возникнет ошибка при загрузке или выполнении программы, переданной exec().

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_03 (master)> ./app
PARENT LOG pid: 46436 grp: 46436, child's pids: 46437, 46438
CHILD №1 LOG pid: 46437, ppid: 46436, grp: 46436
CHILD №2 LOG pid: 46438, ppid: 46436, grp: 46436
app             main.cpp
PARENT LOG child №1 (PID = 46438) has finished with code: 0
  PID TTY           TIME CMD
41398 ttys000    0:00.22 -bash
43805 ttys000    0:01.42 fish
46436 ttys000    0:00.00 ./app
46438 ttys000    0:00.00 (ls)
PARENT LOG child №2 (PID = 46437) has finished with code: 0
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_03 (master)>
```

Рис. 3: Демонстрация работы программы (задание №3).

Задание №4

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. Текст программы приведён на листинге 4.

Листинг 4: Программные каналы

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "string"
#define ERROR_FORK 1
#define ERROR_EXEC 2
#define ERROR_PIPE 3
#define OK 0
int main() {
    int fd[2];
    const char *messages[2] = { "msg\n", "msg msg\n"};
    if (pipe(fd) == -1) {
        return ERROR_PIPE;
    }
    int childpids[2];
    for (int i = 0; i < 2; i++) {
        int pid = fork();
        if (pid == -1) {
            return ERROR_FORK;
        }

        if (pid == 0) {
            close(fd[0]);
            write(fd[1], messages[i], strlen(messages[i]));
            printf("CHILD    %d (pid: %d, ppid: %d, grp: %d) sent message to
                parent\n", i + 1, getpid(), getppid(), getpgrp());
            return OK;
        }
        childpids[i] = pid;
    }
    printf("PARENT LOG pid: %d grp: %d, child's pids: %d, %d\n", getpid(),
       getpgrp(), childpids[0], childpids[1]);
    for (int i = 0; i < 2; i++) {
        int status;
        pid_t childpid = wait(&status);
        if (WIFEXITED(status)) {
            printf("PARENT LOG child    %d (PID = %d) has finished with code
               : %d\n", i + 1, childpid, WEXITSTATUS(status));
```

```
37          } else if (WIFSIGNALED(status)) {
38              printf("PARENT LOG child    %d (PID = %d) has finished because
                    of signal: %d\n", i + 1, childpid, WTERMSIG(status));
39          } else if (WIFSTOPPED(status)) {
40              printf("PARENT LOG child    %d (PID = %d) has been stopped
                    because of signal: %d\n", i + 1, childpid, WSTOPSIG(status));
41          }
42      }
43      char buf[15];
44      close(fd[1]);
45      read(fd[0], buf, 15);
46      printf("PARENT LOG received messages:\n%s", buf);
47      return OK;
48  }
```

Результат работы программы приведён на рисунке ??.

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_04 (master)> ./app
PARENT LOG pid: 47458 grp: 47458, child's pids: 47459, 47460
CHILD №1 (pid: 47459, ppid: 47458, grp: 47458) sent message to parent
CHILD №2 (pid: 47460, ppid: 47458, grp: 47458) sent message to parent
PARENT LOG child №1 (PID = 47460) has finished with code: 0
PARENT LOG child №2 (PID = 47459) has finished with code: 0
PARENT LOG received messages:
msg
msg msg
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_04 (master)>
```

Рис. 4: Демонстрация работы программы (задание №4).

Задание №5

Предок и потомки 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. Вывод соответствующих сообщений на экран. Текст программы приведён на листинге 5.

Листинг 5: Использование сигналов

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "string"
#define ERROR_FORK 1
#define ERROR_EXEC 2
#define ERROR_PIPE 3
#define OK 0
bool sendSig = 0;
void empty(int sig){ }
void sendSigSwitch(int sig) {
    sendSig = 1;
}
int main() {
    signal(SIGINT, empty);
    int fd[2];
    const char *messages[2] = { "msg\n", "msg msg\n"};
    if (pipe(fd) == -1) {
        return ERROR_PIPE;
    }
    int childpids[2];
    for (int i = 0; i < 2; i++) {
        int pid = fork();
        if (pid == -1) {
            return ERROR_FORK;
        }
        if (pid == 0) {
            signal(SIGINT, sendSigSwitch);
            sleep(4);
            if (sendSig) {
                close(fd[0]);
                write(fd[1], messages[i], strlen(messages[i]));
                printf("CHILD    %d (pid: %d, ppid: %d, grp: %d) sent
                   message to parent\n", i + 1, getpid(), getppid(), getpgrp
                   ());
            }
            else {
                printf("No signal sent!\n");
```

```
38              }
39              return OK;
40          }
41          childpids[i] = pid;
42      }
43      printf("PARENT LOG pid: %d grp: %d, child's pids: %d, %d\n", getpid(),
           getpgrp(), childpids[0], childpids[1]);
44      for (int i = 0; i < 2; i++) {
45          int status;
46          pid_t childpid = wait(&status);
47          if (WIFEXITED(status)) {
48              printf("PARENT LOG child    %d (PID = %d) has finished with code
                   : %d\n", i + 1, childpid, WEXITSTATUS(status));
49          }
50          else if (WIFSIGNALED(status)) {
51              printf("PARENT LOG child    %d (PID = %d) has finished because
                   of signal: %d\n", i + 1, childpid, WTERMSIG(status));
52          }
53          else if (WIFSTOPPED(status)) {
54              printf("PARENT LOG child    %d (PID = %d) has been stopped
                   because of signal: %d\n", i + 1, childpid, WSTOPSIG(status));
55          }
56      }
57      char buf[15];
58      close(fd[1]);
59      read(fd[0], buf, 15);
60      printf("PARENT LOG received messages:\n%s", buf);
61      return OK;
62  }
```

Результат работы программы приведён на рисунке 5. При первом запуске был испущен сигнал SIGINT, при втором не был.

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_05 (master)> ./app
PARENT LOG pid: 49353 grp: 49353, child's pids: 49354, 49355
^CCHILD №2 (pid: 49355, ppid: 49353, grp: 49353) sent message to parent
CHILD №1 (pid: 49354, ppid: 49353, grp: 49353) sent message to parent
PARENT LOG child №1 (PID = 49355) has finished with code: 0
PARENT LOG child №2 (PID = 49354) has finished with code: 0
PARENT LOG received messages:
msg msg
msg
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_05 (master)> ./app
PARENT LOG pid: 49364 grp: 49364, child's pids: 49365, 49366
No signal sent!
No signal sent!
PARENT LOG child №1 (PID = 49365) has finished with code: 0
PARENT LOG child №2 (PID = 49366) has finished with code: 0
PARENT LOG received messages:
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/lab_04_05 (master)>
```

Рис. 5: Демонстрация работы программы (задание №5).