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

Дисциплина: Операционные системы

Тема: Процессы. Системные вызовы

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Рязанова Н. Ю.

Москва, 2020 г.

Оглавление

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.

Код программы.

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

int main()
{
	//Потомок 1
	 pid_t childpid1 = fork();

	if (childpid1 == -1) {
		perror("Can`t fork");
		exit(1);
	}
	else if (childpid1 == 0) {
		printf("Before sleep. Child1: pid = %d, ppid = %d, pgrp = %d\n",
\ 		getpid(), getppid(), getpgrp());
		sleep(3);
		printf("After sleep. Child1: pid = %d, ppid = %d, pgrp = %d\n",
		getpid(), getppid(), getpgrp());
		exit(0);
	}

	//Потомок 2
	pid_t childpid2 = fork();

	if (childpid2 == -1) {
		perror("Can`t fork");
```

```
            exit(1);
        }
        else if (childpid2 == 0) {
            printf("Before sleep. Child2: pid = %d, ppid = %d, pgrp = %d\n",
                                        getpid(), getppid(), getpgrp());
            sleep(3);
            printf("After sleep. Child2: pid = %d, ppid = %d, pgrp = %d\n",
                                        getpid(), getppid(), getpgrp());
            exit(0);
        }

        //Предок
        printf("Parent: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n\n",
                    getpid(), getpgrp(), childpid1, childpid2);
        return 0;
}
```

Результат работы программы.

```
Parent: pid = 13906, pgrp = 13906, childpid1 = 13907, childpid2 = 13908

Before sleep. Child1: pid = 13907, ppid = 13906, pgrp = 13906
Before sleep. Child2: pid = 13908, ppid = 13906, pgrp = 13906
main@Asus:~/Desktop/BMSTU/5sem/OS/lab4$ After sleep. Child1: pid = 13907, ppid = 1710, pgrp = 13906
After sleep. Child2: pid = 13908, ppid = 1710, pgrp = 13906
```

Отметим, что при завершении процесса-предка оба потомка продолжили выполнение и получили идентификатор процесса-посредника (ppid = 1710).

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.

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

void waitStatus()
{
	int status;
	pid_t childpid = wait(&status);

	if (WIFEXITED(status)) {
		printf("Child %d exited with code: %d\n", childpid, WEXITSTATUS(status));
	}
	else if (WIFSIGNALED(status)) {
		printf("Child %d ended on signal: %d\n", childpid, WTERMSIG(status));
	}
	else if (WIFSTOPPED(status)) {
		printf("Child %d stopped on signal: %d\n", childpid, WSTOPSIG(status));
	}
}

int main()
{
	//Потомок 1
	pid_t childpid1 = fork();

	if (childpid1 == -1) {
		perror("Can`t fork");
		exit(1);
	}
```

```
        else if (childpid1 == 0) {
                printf("Child1: pid = %d, ppid = %d, pgrp = %d\n",
                                getpid(), getppid(), getpgrp());
                exit(0);
        }

        //Потомок 2
        pid_t childpid2 = fork();

        if (childpid2 == -1) {
                perror("Can`t fork");
                exit(1);
        }
        else if (childpid2 == 0) {
                printf("Child2: pid = %d, ppid = %d, pgrp = %d\n",
                                getpid(), getppid(), getpgrp());
                exit(0);
        }

        //Предок
        printf("Parent: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n\n",
                        getpid(), getpgrp(), childpid1, childpid2);
        waitStatus();
        waitStatus();
        return 0;
}
```

Результат работы программы.

```
Parent: pid = 6402, pgrp = 6402, childpid1 = 6403, childpid2 = 6404

Child1: pid = 6403, ppid = 6402, pgrp = 6402
Child2: pid = 6404, ppid = 6402, pgrp = 6402
Child 6403 exited with code: 0
Child 6404 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.

Код программы.

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

const char* fileName = "chprog";

void waitStatus()
{
	int status;
	pid_t childpid = wait(&status);

	if (WIFEXITED(status)) {
		printf("Child %d exited with code: %d\n", childpid, WEXITSTATUS(status));
	}
	else if (WIFSIGNALED(status)) {
		printf("Child %d ended on signal: %d\n", childpid, WTERMSIG(status));
	}
	else if (WIFSTOPPED(status)) {
		printf("Child %d stopped on signal: %d\n", childpid, WSTOPSIG(status));
	}
}

int main()
{
	//Потомок 1
	pid_t childpid1 = fork();

	if (childpid1 == -1) {
		perror("Can`t fork");
		exit(1);
```

```
        }
        else if (childpid1 == 0) {
                printf("Child1: pid = %d, ppid = %d, pgrp = %d\n",
                                getpid(), getppid(), getpgrp());

                if (execl("/bin/ls", "ls", "-F", NULL) == -1) {
                        perror("Can`t exec");
                        exit(1);
                 }
        }

        // Потомок 2
        pid_t childpid2 = fork();

        if (childpid2 == -1) {
                perror("Can`t fork");
                exit(1);
        }
        else if (childpid2 == 0) {
                printf("Child2: pid = %d, ppid = %d, pgrp = %d\n",
                                getpid(), getppid(), getpgrp());

                if (execl(fileName, fileName, "2", NULL) == -1) {
                        perror("Can`t exec");
                        exit(1);
                }
        }

        //Предок
        printf("Parent: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n\n",
                        getpid(), getpgrp(), childpid1, childpid2);
        waitStatus();
        waitStatus();
        return 0;
}
```

Отметим, что в первом процессе-потомке вызовом execl() происходит запуск команды ls с параметром -F – вывод списка файлов и катологов; во втором процессе-потомке вызовом execl() происходит запуск исполняемого файла программы chprog. Программа выводит на экран сообщение и аргумент.

Код программы chprog.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
        printf("This programm run by child %d!\n", atoi(argv[1]));
        return 0;
}
```

Результат работы программы.

```
Parent: pid = 9076, pgrp = 9076, childpid1 = 9077, childpid2 = 9078

Child1: pid = 9077, ppid = 9076, pgrp = 9076
Child2: pid = 9078, ppid = 9076, pgrp = 9076

This programm run by child 2!
Child 9078 exited with code: 0
a.out*  chprog*  chprog.c  p1.c  p2.c  p3.c  p4.c  p5.c
Child 9077 exited with code: 0
```

Задание 4.

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

Код программы.

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

const int sizeMsg = 48;

void waitStatus()
{
	int status;
	pid_t childpid = wait(&status);

	if (WIFEXITED(status)) {
		printf("Child %d exited with code: %d\n", childpid, WEXITSTATUS(status));
	}
	else if (WIFSIGNALED(status)) {
		printf("Child %d ended on signal: %d\n", childpid, WTERMSIG(status));
	}
	else if (WIFSTOPPED(status)) {
		printf("Child %d stopped on signal: %d\n", childpid, WSTOPSIG(status));
	}
}

int main()
{
	int fd[2];
	if (pipe(fd) == -1) {
		perror("Can`t pipe");
		exit(1);
	}
```

```
//Потомок 1
pid_t childpid1 = fork();

if (childpid1 == -1) {
        perror("Can`t fork");
        exit(1);
}
else if (childpid1 == 0) {
        char msg1[] = "Message from first child!\n";

        close(fd[0]);
        write(fd[1], msg1, strlen(msg1));

        exit(0);
}



//Потомок 2
pid_t childpid2 = fork();

if (childpid2 == -1) {
        perror("Can`t fork");
        exit(1);
}
else if (childpid2 == 0) {
        char msg2[] = "Message from child 2!\n";

        close(fd[0]);
        write(fd[1], msg2, strlen(msg2));

        exit(0);
}

//Предок
printf("Parent: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n\n",
                        getpid(), getpgrp(), childpid1, childpid2);
waitStatus();
waitStatus();
```

```
    printf("\nMsgs from pipe:\n");
    char msg[sizeMsg];

    close(fd[1]);
    read(fd[0], msg, sizeMsg);

    printf("%s\n", msg);
    return 0;
}
```

Результат работы программы.

```
Parent: pid = 8648, pgrp = 8648, childpid1 = 8649, childpid2 = 8650

Child 8649 exited with code: 0
Child 8650 exited with code: 0

Msgs from pipe:
Message from first child!
Message from child 2!
```

Задание 5.

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

Код программы.

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

const int sizeMsg = 48;
static bool wasSignal = false;

void waitStatus()
{
	int status;
	pid_t childpid = wait(&status);

	if (WIFEXITED(status)) {
		printf("Child %d exited with code: %d\n", childpid, WEXITSTATUS(status));
	}
	else if (WIFSIGNALED(status)) {
		printf("Child %d ended on signal: %d\n", childpid, WTERMSIG(status));
	}
	else if (WIFSTOPPED(status)) {
		printf("Child %d stopped on signal: %d\n", childpid, WSTOPSIG(status));
	}
}

void catchSignal(int signalNum)
{
	printf("\nCatched signal %d!\n\n", signalNum);
	wasSignal = true;
}
```

```
int main()
{
	int fd[2];
	if (pipe(fd) == -1) {
		perror("Can`t pipe");
		exit(1);
	}

	printf("Press CTRL + C to write in pipe\n");
	signal(SIGINT, catchSignal);
	sleep(4);

	//Потомок 1
	pid_t childpid1 = fork();

	if (childpid1 == -1) {
		perror("Can`t fork");
		exit(1);
	}
	else if (childpid1 == 0) {
		if (wasSignal) {
			char msg1[] = "Message from first child!\n";

			close(fd[0]);
			write(fd[1], msg1, strlen(msg1));
		}
		exit(0);
	}

	//Потомок 2
	pid_t childpid2 = fork();

	if (childpid2 == -1) {
		perror("Can`t fork");
		exit(1);
	}
	else if (childpid2 == 0) {
		if (wasSignal) {
			char msg2[] = "Message from child 2!\n";
```

```
                close(fd[0]);
                write(fd[1], msg2, strlen(msg2));
            }
            exit(0);
        }

        //Предок
        printf("Parent: pid = %d, pgrp = %d, childpid1 = %d, childpid2 = %d\n\n",
                                getpid(), getpgrp(), childpid1, childpid2);
        waitStatus();
        waitStatus();

        if (wasSignal) {
            printf("\nMsgs from pipe:\n");
            char msg[sizeMsg];

            close(fd[1]);
            read(fd[0], msg, sizeMsg);

            printf("%s\n", msg);
        }
        else {
            printf("\nNo signal.\nChildrens didn`t write in pipe!\n");
        }
        return 0;
}
```

Отметим, что процессы-потомки пишут сообщения в канал, если был получен сигнал Ctrl+C. На ожидание сигнала отводится 4 секунды.

Результат работы программы.

Случай 1. Сигнал был отправлен.

```
Press CTRL + C to write in pipe
^C
Catched signal 2!

Parent: pid = 9877, pgrp = 9877, childpid1 = 9878, childpid2 = 9879

Child 9878 exited with code: 0
Child 9879 exited with code: 0

Msgs from pipe:
Message from first child!
Message from child 2!
```

Случай 2. Сигнал не был отправлен.

```
Press CTRL + C to write in pipe
Parent: pid = 9902, pgrp = 9902, childpid1 = 9904, childpid2 = 9905

Child 9904 exited with code: 0
Child 9905 exited with code: 0

No signal.
Childrens didn`t write in pipe!
```