## 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)

**ОТЧЕТ**

**по лабораторной работе №** 4

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

**Дисциплина: Операционные системы**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-56Б  (Группа) | (Подпись, дата) | Ж. Р. Турсунов  (И.О. Фамилия) |
| Преподаватель |  | (Подпись, дата) | Н.Ю. Рязанова  (И.О. Фамилия) |

Москва, 2020

# 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.

# Листинг кода:

#include <stdio.h>

#include <stdlib.h>

int main**()**

**{**

int child1 **=** fork**();**

**if** **(** child1 **==** **-**1 **)**

**{**

perror**(**"couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child1 **==** 0 **)**

**{**

//потомок

sleep**(**1**);**

printf**(** "Child1: pid=%d; group=%d;parent=%d\n"**,** getpid**(),** getpgrp**(),** getppid**());**

**return** 0**;**

**}**

int child2 **=** fork**();**

**if** **(** child2 **==** **-**1 **)**

**{**

perror**(**"couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child2 **==** 0 **)**

**{**

//потомок

sleep**(**1**);**

printf**(** "Child2: pid=%d; group=%d; parent=%d\n"**,** getpid**(),** getpgrp**(),** getppid**()** **);**

**return** 0**;**

**}**

**else**

**{**

//предок

printf**(** "Parent: pid=%d; group=%d; child1=%d; child2=%d\n"**,** getpid**(),** getpgrp**(),** child1**,** child2 **);**

**return** 0**;**

**}**

**}**

# Полученный результат:

# 

# 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.

# Листинг кода:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main**()**

**{**

int child1 **=** fork**();**

**if** **(** child1 **==** **-**1 **)**

**{**

perror**(**"Couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child1 **==** 0 **)**

**{**

//потомок

sleep**(**2**);**

printf**(** "Child1: pid=%d; group=%d; parent=%d\n"**,** getpid**(),** getpgrp**(),** getppid**()** **);**

**}**

int child2 **=** fork**();**

**if** **(** child2 **==** **-**1 **)**

**{**

perror**(**"couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child2 **==** 0 **)**

**{**

//потомок

sleep**(**1**);**

printf**(** "Child2: pid=%d; group=%d; parent=%d\n"**,** getpid**(),** getpgrp**(),** getppid**()** **);**

**return** 0**;**

**}**

**else**

**{**

//предок

printf**(** "Parent: pid=%d; group=%d; child1=%d; child2=%d\n"**,** getpid**(),** getpgrp**(),** child1**,** child2**);**

int status**;**

pid\_t ret\_value**;**

ret\_value **=** wait**(** **&**status **);**

**if** **(** WIFEXITED**(**status**)** **)**

printf**(**"Parent: child %d finished with %d code.\n"**,** ret\_value**,** WEXITSTATUS**(**status**)** **);**

**else** **if** **(** WIFSIGNALED**(**status**)** **)**

printf**(** "Parent: child %d finished from signal with %d code.\n"**,** ret\_value**,** WTERMSIG**(**status**));**

**else** **if** **(** WIFSTOPPED**(**status**)** **)**

printf**(**"Parent: child %d finished from signal with %d code.\n"**,** ret\_value**,** WSTOPSIG**(**status**));**

**return** 0**;**

**}**

**}**

# Полученный результат:

# 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.

# Листинг кода:

#include <stdio.h>

#include <stdlib.h> //exit

#include <unistd.h> //execlp

#include <sys/types.h>

#include <sys/wait.h>

int main**()**

**{**

int child1 **=** fork**();**

**if** **(** child1 **==** **-**1 **)**

**{**

perror**(**"Couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child1 **==** 0 **)**

**{**

//потомок

printf**(** "Child: pid=%d; group=%d; parent=%d\n\n"**,** getpid**(),** getpgrp**(),** getppid**()** **);**

**if** **(** execlp**(**"ps"**,** "ps"**,** "al"**,** 0**)** **==** **-**1 **)**

**{**

perror**(** "Child couldn't exec." **);**

exit**(**1**);**

**}**

**}**

int child2 **=** fork**();**

**if** **(** child2 **==** **-**1 **)**

**{**

perror**(**"Couldn't fork."**);**

exit**(**1**);**

**}**

**if** **(** child2 **==** 0 **)**

**{**

//потомок

printf**(** "Child: pid=%d; group=%d; parent=%d\n\n"**,** getpid**(),** getpgrp**(),** getppid**()** **);**

**if** **(** execlp**(**"/bin/ls"**,** "ls"**,** "-l"**,** 0**)** **==** **-**1 **)**

**{**

perror**(** "Child couldn't exec." **);**

exit**(**1**);**

**}**

**}**

**if** **(**child1 **!=** 0 **&&** child2 **!=** 0**)**

**{**

//предок

printf**(** "Parent: pid=%d; group=%d; child1=%d child2=%d\n"**,** getpid**(),** getpgrp**(),** child1**,** child2 **);**

int status**;**

pid\_t ret\_value**;**

ret\_value **=** wait**(** **&**status **);**

**if** **(** WIFEXITED**(**status**)** **)**

printf**(**"Parent: child %d finished with %d code.\n\n"**,** ret\_value**,** WEXITSTATUS**(**status**)** **);**

**else** **if** **(** WIFSIGNALED**(**status**)** **)**

printf**(** "Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WTERMSIG**(**status**));**

**else** **if** **(** WIFSTOPPED**(**status**)** **)**

printf**(**"Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WSTOPSIG**(**status**));**

ret\_value **=** wait**(** **&**status **);**

**if** **(** WIFEXITED**(**status**)** **)**

printf**(**"Parent: child %d finished with %d code.\n\n"**,** ret\_value**,** WEXITSTATUS**(**status**)** **);**

**else** **if** **(** WIFSIGNALED**(**status**)** **)**

printf**(** "Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WTERMSIG**(**status**));**

**else** **if** **(** WIFSTOPPED**(**status**)** **)**

printf**(**"Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WSTOPSIG**(**status**));**

**return** 0**;**

**}**

**}**

# Полученный результат:

**Задание 4**: Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

# Листинг кода:

#include <stdlib.h> //exit

#include <unistd.h> //pipe

#include <string.h> //strlen

#include <sys/types.h>

#include <sys/wait.h>

int main**()**

**{**

int descr**[**2**];** //дескриптор \_одного\_ программного канала

//[0] - выход для чтения, [1] - выход для записи

//потомок унаследует открытый программный канал предка

**if** **(** pipe**(**descr**)** **==** **-**1**)**

**{**

perror**(** "Couldn't pipe." **);**

exit**(**1**);**

**}**

int child1 **=** fork**();**

**if** **(** child1 **==** **-**1 **)**

**{**

perror**(** "Couldn't fork." **);**

exit**(**1**);**

**}**

**if** **(** child1 **==** 0 **)**

**{**

//потомок

close**(** descr**[**0**]** **);** //потомок ничего не запишет в канал

char msg1**[]** **=** "hello Solar system "**;**

//считывание из программного канала по 1 символу

write**(**descr**[**1**],** msg1**,** 64**);**

exit**(**0**);**

**}**

int child2 **=** fork**();**

**if** **(** child2 **==** **-**1 **)**

**{**

perror**(** "Couldn't fork." **);**

exit**(**1**);**

**}**

**if** **(** child2 **==** 0 **)**

**{**

//потомок

close**(** descr**[**0**]** **);** //потомок ничего не запишет в канал

char msg2**[]** **=** "hello Earth "**;**

//считывание из программного канала по 1 символу

write**(**descr**[**1**],** msg2**,** 64**);**

exit**(**0**);**

**}**

**if** **(**child1 **!=** 0 **&&** child2 **!=** 0**)**

**{**

//предкод

close**(** descr**[**1**]** **);** //предок ничего не считает из канала

char msg1**[**64**];**

read**(** descr**[**0**],** msg1**,** 64**);** //передаём сообщение в канал

char msg2**[**64**];**

read**(** descr**[**0**],** msg2**,** 64**);** //передаём сообщение в канал

printf**(**"Parent: reads %s %s"**,** msg1**,** msg2**);**

int status**;**

pid\_t ret\_value**;**

ret\_value **=** wait**(** **&**status **);**

**if** **(** WIFEXITED**(**status**)** **)**

printf**(**"Parent: child %d finished with %d code.\n\n"**,** ret\_value**,** WEXITSTATUS**(**status**)** **);**

**else** **if** **(** WIFSIGNALED**(**status**)** **)**

printf**(** "Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WTERMSIG**(**status**));**

**else** **if** **(** WIFSTOPPED**(**status**)** **)**

printf**(**"Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WSTOPSIG**(**status**));**

ret\_value **=** wait**(** **&**status **);**

**if** **(** WIFEXITED**(**status**)** **)**

printf**(**"Parent: child %d finished with %d code.\n\n"**,** ret\_value**,** WEXITSTATUS**(**status**)** **);**

**else** **if** **(** WIFSIGNALED**(**status**)** **)**

printf**(** "Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WTERMSIG**(**status**));**

**else** **if** **(** WIFSTOPPED**(**status**)** **)**

printf**(**"Parent: child %d finished from signal with %d code.\n\n"**,** ret\_value**,** WSTOPSIG**(**status**));**

**}**

**return** 0**;**

**}**

# Полученный результат:

# 

**Задание 5:** В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

# Листинг кода:

#include <stdio.h> //printf

#include <stdlib.h> //exi

#include <unistd.h> //pipe

#include <string.h> //strlen

#include <signal.h>

#include <time.h>

int parent\_flag **=** 0**;**

void sigint\_catcher**(**int signum**){**

printf**(** "\nProccess Catched signal #%d\n"**,** signum**);**

printf**(**"Sent secret email to son!\n"**);**

parent\_flag **=** 1**;**

**}**

int main**()**

**{**

int child**;**

int descr**[**2**];** //дескриптор \_одного\_ программного канала

signal**(**SIGINT**,** sigint\_catcher**);**

**if** **(** pipe**(**descr**)** **==** **-**1**){**

perror**(** "Couldn't pipe." **);**

exit**(**1**);**

**}**

child **=** fork**();**

**if** **(** child **==** **-**1 **){**

perror**(** "Couldn't fork." **);**

exit**(**1**);**

**}**

**if** **(** child **==** 0 **){**

close**(** descr**[**1**]** **);** //потомок ничего не запишет в канал

char msg**[**64**];**

memset**(** msg**,** 0**,** 64 **);**

int i **=** 0**;**

**while(** read**(**descr**[**0**],** **&(**msg**[**i**++]),** 1**)** **!=** '\0' **)** **;**

printf**(**"Child: reading..\n\n"**);**

printf**(** "Child: read <%s>\n"**,** msg **);**

**}**

**else{**

close**(** descr**[**0**]** **);** //предок ничего не считает из канала

printf**(** "Parent: waiting for CTRL+C signal for 3 seconds...\n" **);**

sleep**(**3**);**

**if** **(**parent\_flag**){**

char msg**[**64**]** **=** "It`s my secret email for you, son. Father."**;**

write**(** descr**[**1**],** msg**,** strlen**(**msg**)** **);** //передаём сообщение в канал

exit**(**0**);**

**}**

**else{**

char msg**[**64**]** **=** "Hello my child!"**;**

write**(** descr**[**1**],** msg**,** strlen**(**msg**)** **);** //передаём сообщение в канал

**}**

**return** 0**;**

**}**

**}**

# Полученный результат:

# С сигналом

# 

# Без сигнала

# 