**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине

«Операционные системы»

«Разработка программ обработки файлов в ОС Windows»

Вариант 2

Студент

Группа АИ-20-1 \_\_\_\_\_\_\_\_\_\_ Глубоков Г.В.

Подпись, дата

Руководитель

к.т.н.,доц. Батищев Р. В.

Подпись, дата

Липецк 2022г.

# Задание кафедры:

Написать программу, запускающую несколько внешних программ (порождающую несколько процессов) в определенном порядке, а также создающую наследуемые дескрипторы файлов исходного процесса и перенаправляющую дескрипторы ввода/вывода первых трех дочерних процессов на файлы исходного процесса. При этом число процессов и порядок их создания, способ перенаправления (P: 1 – изменения в STARTUPINFO, 2 – с помощью *GetStdHandle*/*SetStdHandle*), порядок перенаправления (Q: 1 – ввод, вывод, ввод/вывод, 2 – вывод, ввод/вывод, ввод, 3 – ввод/вывод, ввод, вывод) выбираются из табл. 2 прил.).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № вар. | Число процессов | Порядок создания процессов | P | Q |
| 2 | 3 | Вначале параллельно первый и второй, после того, как хотя бы один из них завершится, – третий | 2 | 2 |

# Теоретические сведения:

Системный вызов **fork()** позволяет одному процессу, родителю, создавать новый, дочерний процесс. Оба этих процесса являются (почти) идентичными: потомок получает копии родительского стека, данных, кучи, копии родительских сегментов стека Х и текста.

Системный вызов **wait**(&status) имеет два назначения. Во-первых, если работа потомка текущего процесса еще не была завершена путем вызова exit(), функция **wait()** приостанавливает выполнение родителя, пока не будет завершен один из его потомков. Во-вторых, код завершения потомка возвращается через аргумент функции **wait()**.Изменение файлового смещения:Системный вызов **execl(pathname, argv)** загружает в память процесса новую программу (расположенную в pathname, с аргументами argv). Текст существующей программы сбрасывается, а для новой программы заново создаются сегменты со стеком, данными и кучей. Эту операцию часто называют выполнением новой программы.

Функция **system()** позволяет вызывающей программе выполнять произвольные консольные команды.

Ход работы:

Код программы на C++:

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

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <iostream>

int main(){

pid\_t pid1 = fork();//создаём процесс

if(pid1 == 0){

printf("Forked!\n");

int pf1 = open("proc1.txt", O\_RDWR); //вот сюда перенаправим вывод

system("cd /Users/german/Desktop/OC/Laba-2");//вызовем какую-нибудь стандартную утилиту

system("g++ proc1.cpp -o proc1");

dup2(pf1, STDOUT\_FILENO);//клонируем файловый дескриптор, теперь он открыт одновременно в pf и stdout

execl("./proc1"," ","first", NULL);

}else if(pid1 < 0){

fprintf(stderr, "Some error on fork\n");

return -1;

}

pid\_t pid2 = fork();//создаём процесс

if(pid2 == 0){

printf("Forked!\n");

system("cd /Users/german/Desktop/OC/Laba-2");//bash

int pf2 = open("proc2.txt", O\_RDWR);

dup2(pf2, STDOUT\_FILENO);//перенаправляем вывод

int pf3 = open("a.txt", O\_RDONLY);

dup2(pf3, STDIN\_FILENO);//перенаправляем ввод

system("g++ proc2.cpp -o proc2");

execl("./proc2"," ","second", NULL);

}else if(pid2 < 0){

fprintf(stderr, "Some error on fork\n");

return -1;

}

printf("Success!\n");

wait(NULL);//дожидаемся завершения

wait(NULL);//дожидаемся завершения

pid\_t pid3 = fork();//создаём процесс

if(pid3 == 0){

printf("Forked!\n");

system("cd /Users/german/Desktop/OC/Laba-2"); //bash

int pf4 = open("proc2.txt", O\_RDWR);

int pf3 = open("a.txt", O\_RDONLY);

dup2(pf3, STDIN\_FILENO); //перенаправляем ввод

system("g++ proc2.cpp -o proc3"); //создаём файл out

execl("./proc3"," ","third", NULL);

}else if(pid3 < 0){

fprintf(stderr, "Some error on fork\n");

return -1;

}

wait(NULL);

printf("Done\n");

}

## Пример выполнения:

На рисунке 1 показаны два первых параллельных процесса и их pid

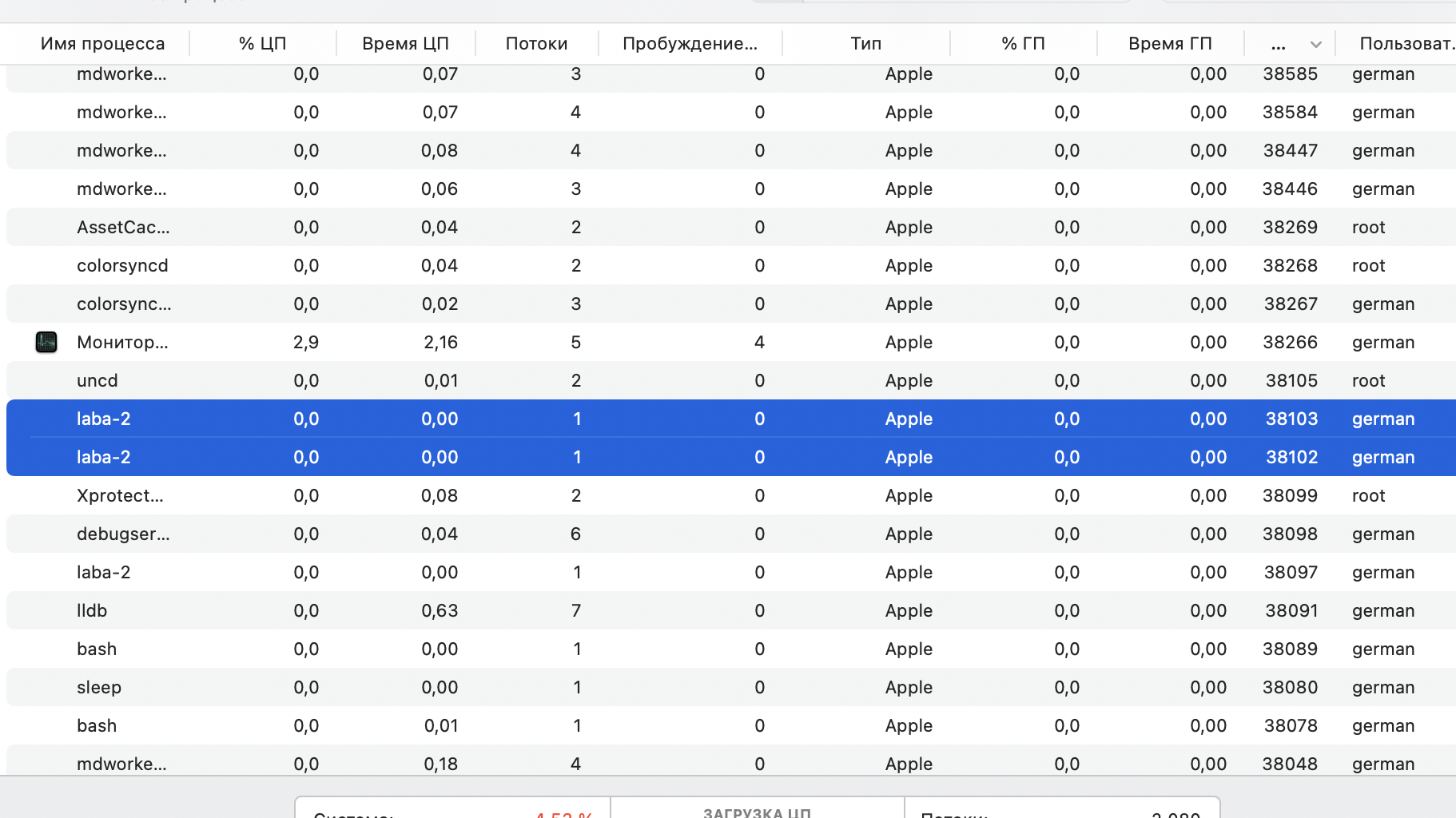
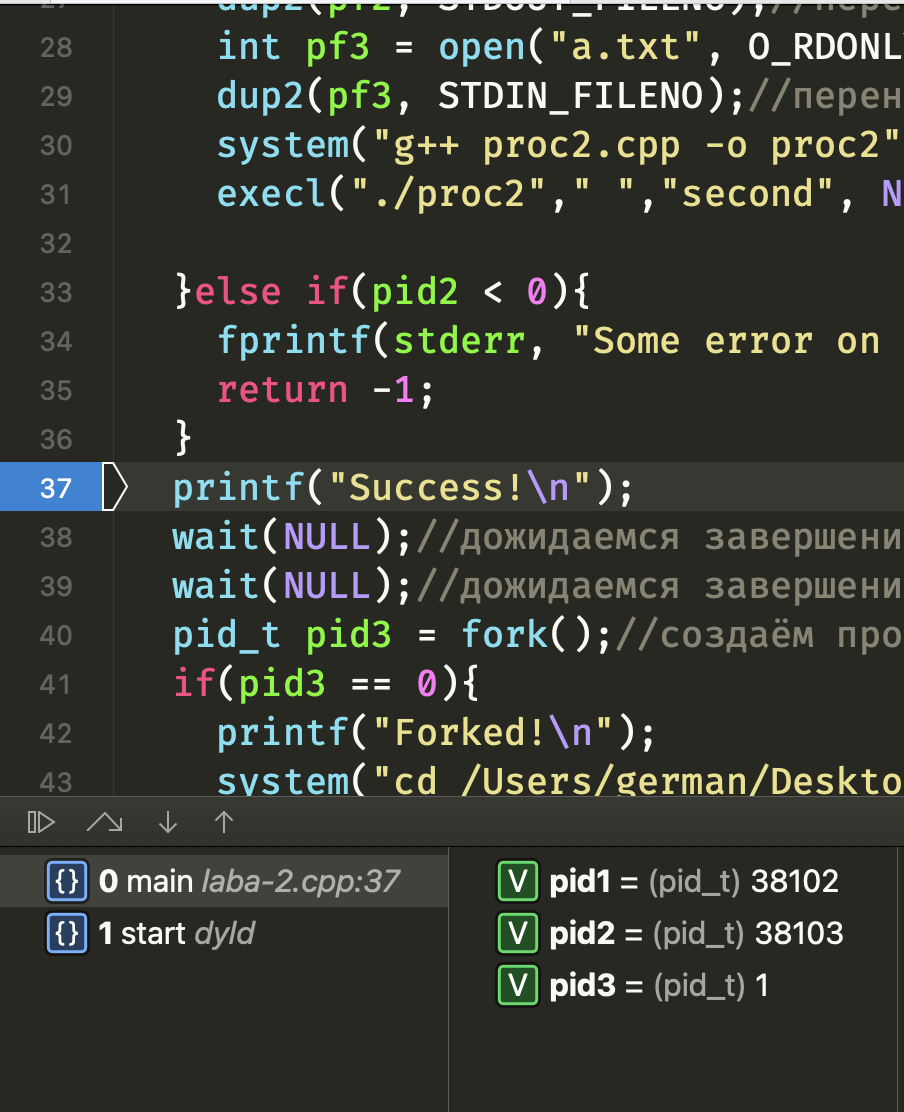


Рисунок 1 – Процессы