МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчёт по лабораторной работе № 22**

по дисциплине: ”Системное программирование”

на тему: ***”Многозадачность процессов и этапы компиляции”***

Вариант 11

Выполнил**:** студент группы 10701222 Хасаншин А.В

Демянов Л.С

Принял**:** пр. Давыденко Н.В.

# Лабораторная работа № 22. Многозадачность процессов и этапы компиляции

## Цель: Изучить встроенный инструментарий для разработки приложений под семейство ОС Linux и фундаментальные основы системного программирования с использованием компиляторов gcc/g++, отладчика gdb и других для проектирования, компиляции, отладки и запуска приложений на языке программирования С/C++.

## Изучить базовую концепцию операционной системы, «Процесс». Получить представление о многозадачности процессов.

## Задание 1.

Используя материалы (исходные файлы) упражнения №1 разработайте про-

грамму, в которой дочерний процесс запускается системным вызовом fork в

цикле, ограниченном счетчиком проходов по циклу. Через пять секунд роди-

тельский процесс должен закончить работу, а дочерний процесс выдать ко-

манду ps. Внимание. Список выполняющихся процессов можно получить с

помощью команды $ps.

Теперь нужно осознать, как осуществляется взаимодействие процессов. Стан-

дартным образом для этой цели используется канал (pipe). Но сейчас мы осуще-

ствим взаимодействие через текстовый файл. Создайте программу, в которой

один процесс будет писать в файл какую-то строку, а второй будет считывать.

Проведите компиляцию однофайлового проекта помощью скрипты bash с

прохождением всех стадий компиляции.

Сборка проекта должна содержать файлы с результатами препроцессинга.

Исследуйте фалы препроцессора, найдите в них код своей программы.

Определите размеры исходных, препроцессорных, ассемблерных, объектных

и исполняемых файлов. С помощью соответствующей консольной команды

или с помощь проводника определите форматы этих файлов. Результаты

подтвердите скриншотами. Сделайте краткий вывод по выполненной работе.

### Скриншоты результатов

Используя материалы (исходные файлы) упражнения №1 разработайте про-

грамму, в которой дочерний процесс запускается системным вызовом fork в

цикле, ограниченном счетчиком проходов по циклу. Через пять секунд роди-

тельский процесс должен закончить работу, а дочерний процесс выдать ко-

манду ps. Внимание. Список выполняющихся процессов можно получить с

помощью команды $ps.

#include <stdio.h>

#include <unistd.h> // для fork, sleep, execlp

#include <sys/types.h> // для pid\_t

#include <stdlib.h> // для exit

#include <sys/wait.h> // для wait

int main() {

int count = 5; // Количество итераций цикла (количество дочерних процессов)

pid\_t pid;

for (int i = 0; i < count; i++) {

pid = fork(); // Создаем дочерний процесс

if (pid == -1) {

// Ошибка при создании процесса

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Это код дочернего процесса

sleep(5); // Задержка на 5 секунд, чтобы дать родительскому процессу завершиться

execlp("ps", "ps", NULL); // Замена текущего процесса на выполнение команды ps

perror("execlp"); // Если exec не сработал, выводим ошибку

exit(EXIT\_FAILURE);

}

// Родительский процесс продолжает выполнение и создает следующий дочерний процесс

}

// Родительский процесс ждет 5 секунд и завершает свою работу

sleep(5);

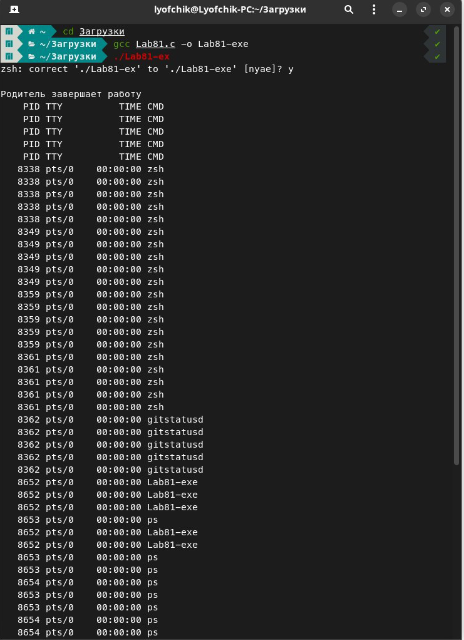
printf("Родитель завершает работу\n");

// Ожидание завершения всех дочерних процессов, чтобы избежать зомби-процессов

while (wait(NULL) > 0);

return 0;

}



Теперь нужно осознать, как осуществляется взаимодействие процессов. Стан-

дартным образом для этой цели используется канал (pipe). Но сейчас мы осуще-

ствим взаимодействие через текстовый файл. Создайте программу, в которой

один процесс будет писать в файл какую-то строку, а второй будет считывать.

Проведите компиляцию однофайлового проекта помощью скрипты bash с

прохождением всех стадий компиляции.

Сборка проекта должна содержать файлы с результатами препроцессинга.

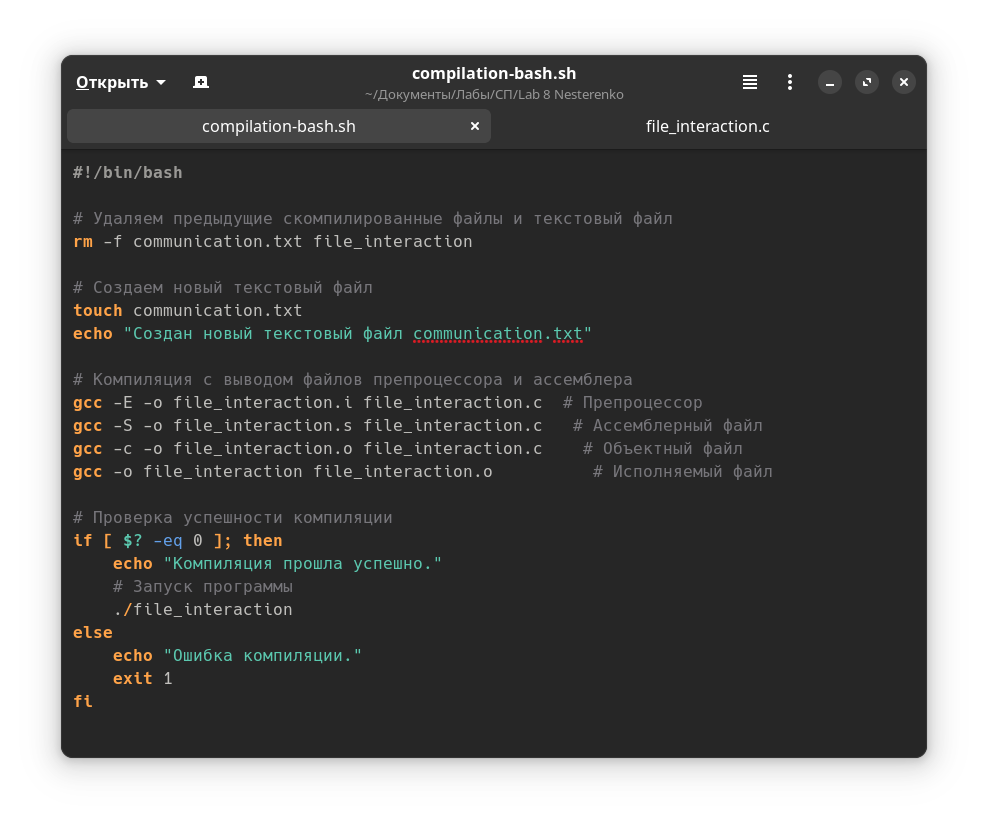
Исследуйте фалы препроцессора, найдите в них код своей программы.

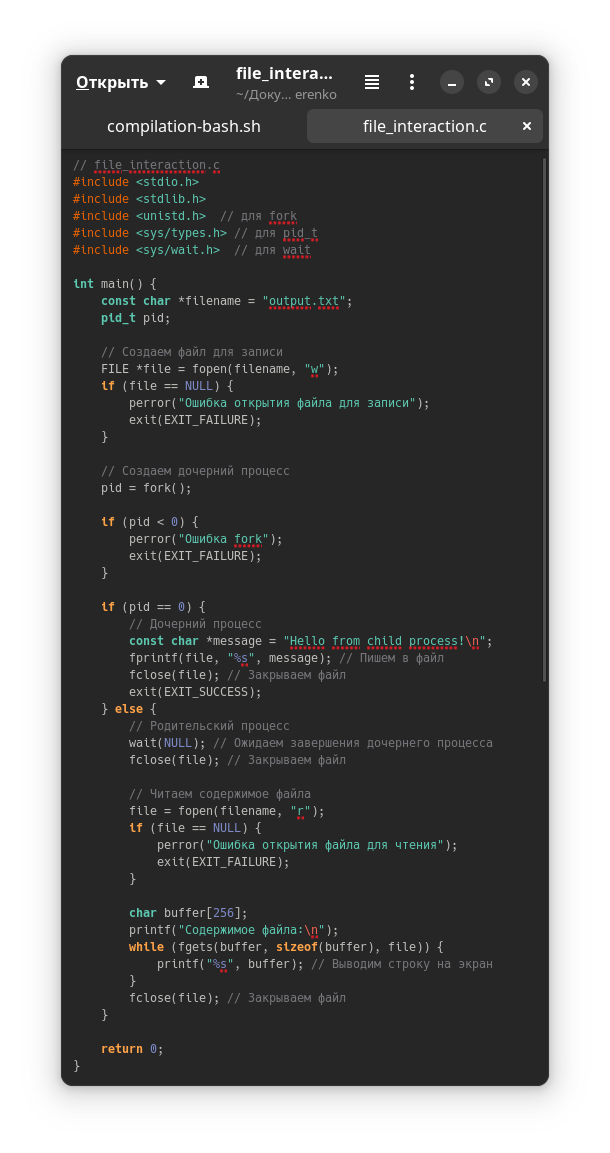
Определите размеры исходных, препроцессорных, ассемблерных, объектных

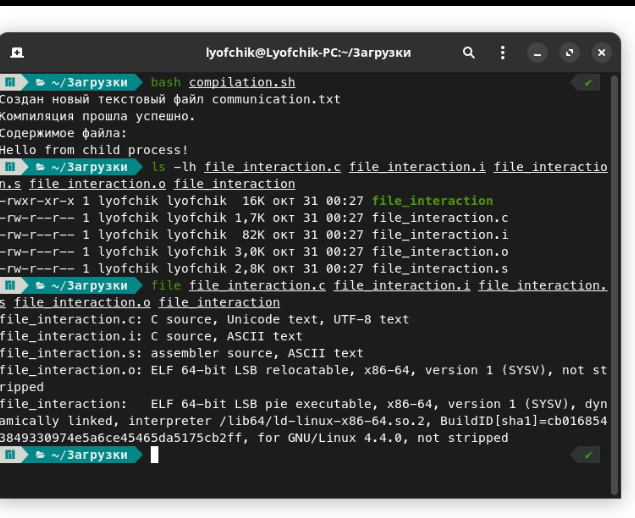
и исполняемых файлов. С помощью соответствующей консольной команды

или с помощь проводника определите форматы этих файлов. Результаты

подтвердите скриншотами. Сделайте краткий вывод по выполненной работе.







## Задание 2.

### Скриншоты результатов

## Задание 3.

Создайте программу запускающую «микросервисы»: программу-календарь, браузер, проводник, текстовый редактор, установленные в вашей системе. С помощью очень полезной утилиты strace, запустите переданную ей вами разработанную программу и выведите в стандартный поток ошибок отчет об использованных системных вызовах.

**Код Программы:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr, "Usage: %s <program> [args]\n", argv[0]);

return 1;

}

// Запуск программы с аргументами

execvp(argv[1], argv + 1);

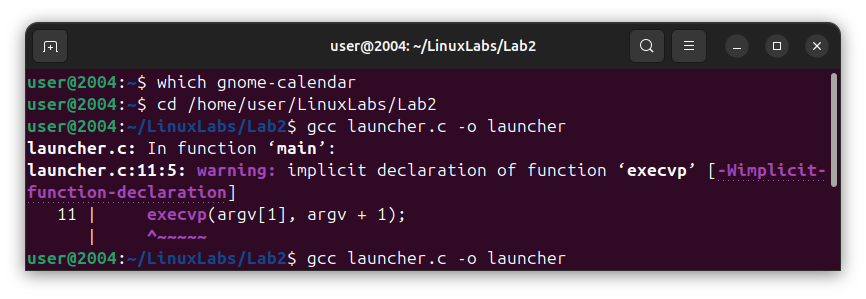
// Если execvp вернулась, значит произошла ошибка

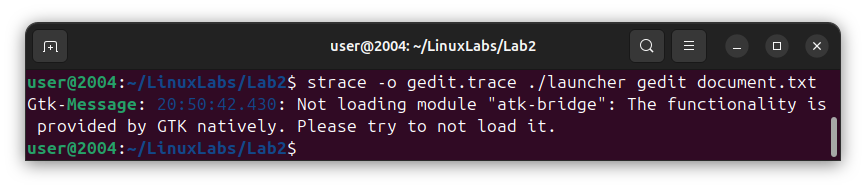
perror("execvp");

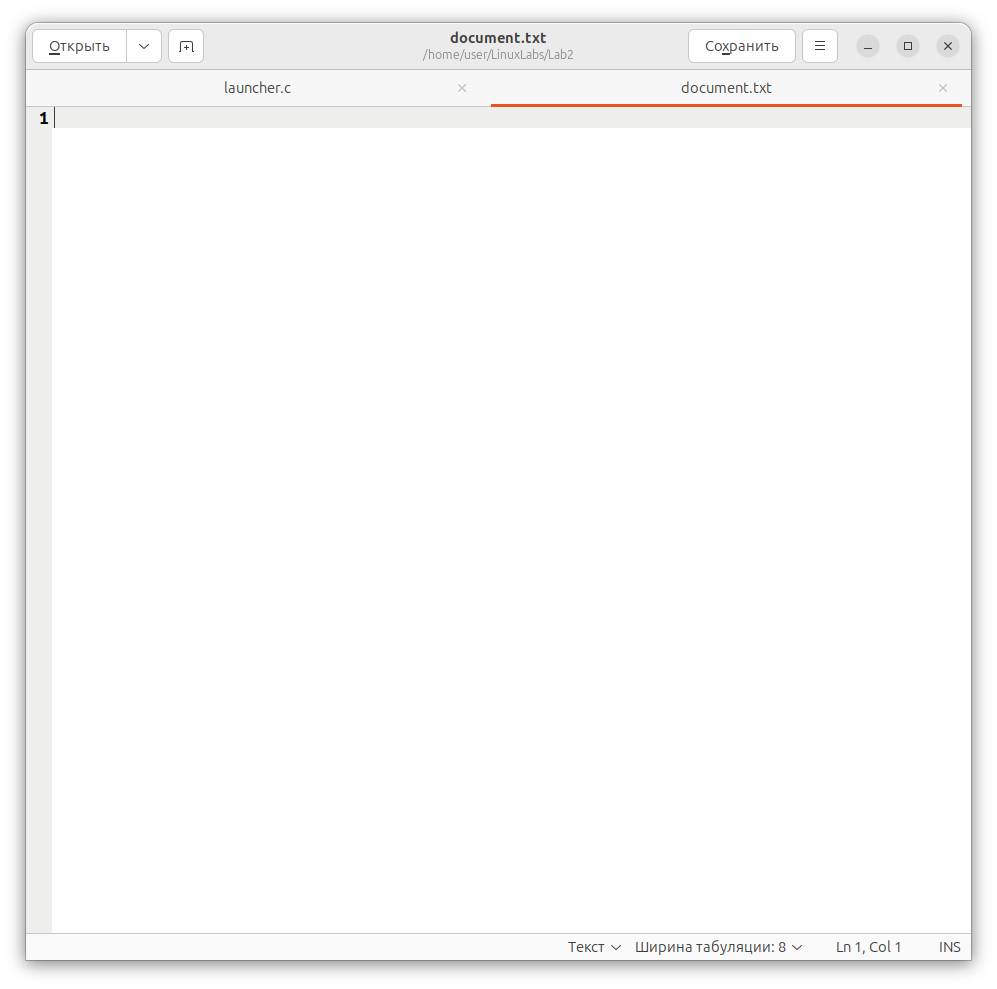
return 1;

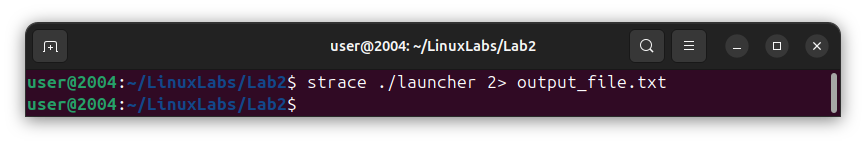
}

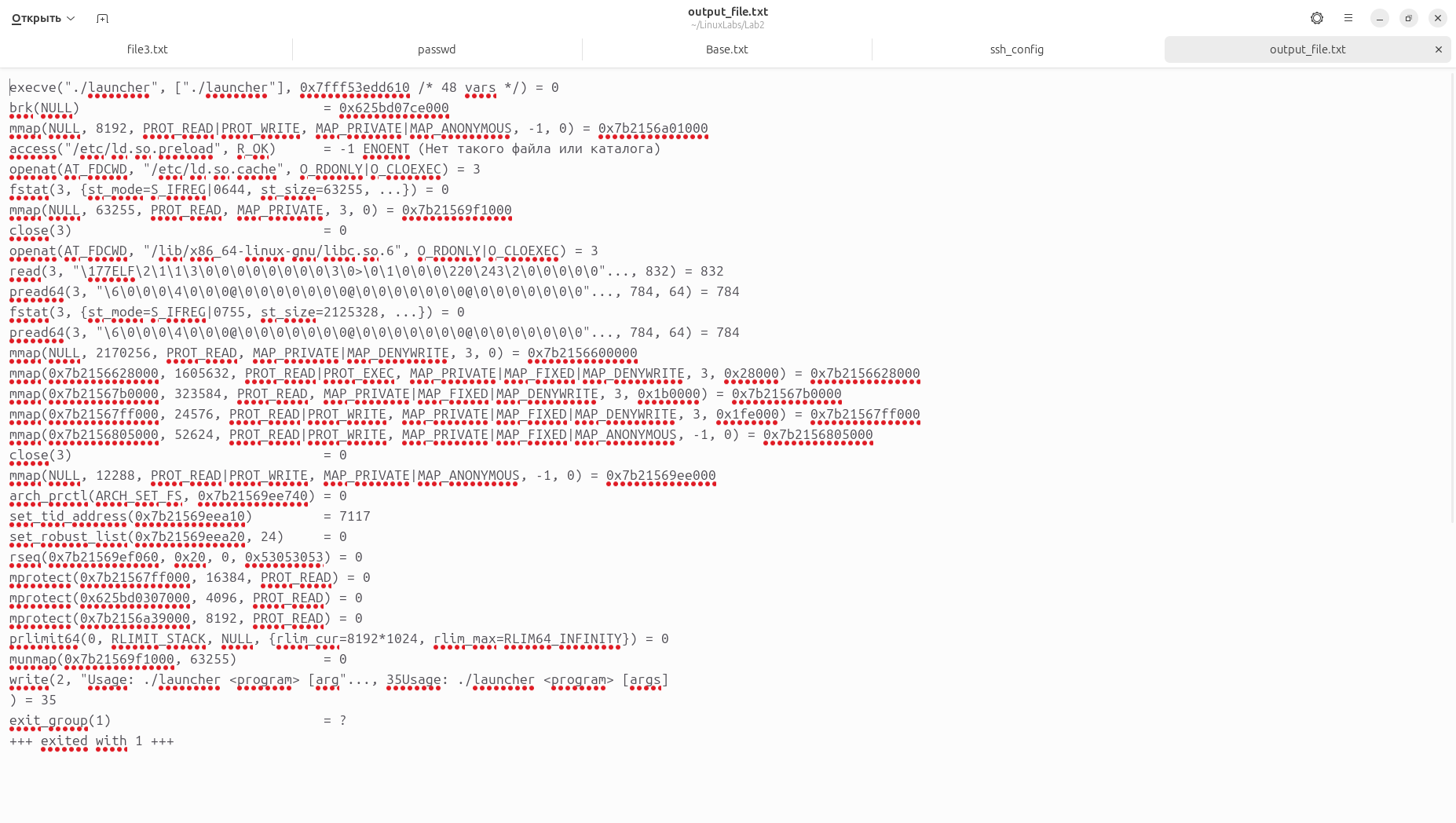
**Результаты\_работы:**











### Скриншоты результатов

## Контрольные вопросы

Что такое автосборка? Какие приемущества она дает?

Автосборка — это автоматизированный процесс компиляции и сборки программного обеспечения, который не требует ручного вмешательства.

Преимущества автосборки:

1. Снижение ошибок: Меньше человеческих ошибок в процессе сборки.

2. Скорость разработки: Разработчики сосредотачиваются на коде, а не на сборке.

3. Стандартизация: Обеспечивает единообразие в процессах сборки.

4. Тестирование: Легко интегрировать автоматические тесты.

5. Развертывание: Упрощает упаковку и развертывание приложений.

6. Обратная связь: Быстрая обратная связь о состоянии сборки и тестирования.

7. Гибкость: Легко адаптируется к изменениям в проекте.

8. Снижение затрат: Экономия ресурсов благодаря меньшему времени на сборку.

Примеры инструментов:

- Make

- CMake

- Gradle

- Maven

- Jenkins

Автосборка повышает продуктивность и качество разработки.