

- Ты умрешь таким же, каким жил:  
жалким и невежественным.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
“Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ  
И КОМПЬЮТЕРНОЙ ТЕХНИКИ

### ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине  
“ОПЕРАЦИОННЫЕ СИСТЕМЫ”



выполнил:  
Студент группы Р33314  
**Птицын Максим Евгеньевич**  
Преподаватель  
**Клименков Сергей Викторович**

2023 г.  
г. Санкт-Петербург

- ① Ты умрёшь таким же, каким жил:  
жалким и невежественным.

## Содержание

1 Github:

3

2 Вывод:

3



- Ты умрешь таким же, каким жил:  
жалким и невежественным.

## 1 Github:

[github.com/Kyoto67/xv6-riscv](https://github.com/Kyoto67/xv6-riscv)

## 2 Вывод:

В ходе выполнения курсовой работы были выполнены следующие задания, связанные с операционной системой xv6:

1. \*\*Работа с Unix Pipes и программой pingpong.c:\*\* Для начала, мы ознакомились с основами операционной системы xv6, а также изучили её системные вызовы, включая 'read', 'fork', 'read', 'write', и 'getpid'. Затем мы разработали программу 'pingpong.c', которая успешно выполняет следующие шаги:

- Создание пайпа с использованием системного вызова 'pipe'. - Создание дочернего процесса с использованием системного вызова 'fork'. - Отправка сообщения "ping" из родительского процесса в дочерний процесс с использованием системного вызова 'write'. - Чтение сообщения "ping" в дочернем процессе с использованием системного вызова 'read', вывод сообщения "<child pid>: got ping", и отправка сообщения "pong" обратно в родительский процесс. - Чтение сообщения "pong" в родительском процессе с использованием системного вызова 'read', вывод сообщения "<parent pid>: got pong".

Это задание дало нам практический опыт работы с Unix pipes, процессами и системными вызовами в xv6.

2. \*\*Реализация системного вызова dump:\*\* Мы добавили новый системный вызов 'dump', который позволяет выводить на экран состояние регистров 's2' - 's12' вызывающего процесса. Этот вызов включал следующие этапы:

- Добавление объявления функции 'dump' в файл 'user/user.h'. - Внесение изменений в файл 'user/usys.pl' для генерации ассемблерных инструкций. - Реализация функции 'dump' в файле 'kernel/proc.c', которая выводит значения указанных регистров текущего процесса, учитывая ограничения младшей 32-битной части каждого регистра. - Редактирование файлов 'kernel/syscall.h', 'kernel/sysproc.c', и 'kernel/syscall.c' для добавления возможности вызывать 'dump' из user-space.

Мы также успешно запустили утилиту 'dumptests', которая автоматически сравнила результаты нашего системного вызова 'dump' с фактическими значениями регистров. Этот этап дал нам понимание работы с системными вызовами и работой с регистрами процесса.

3. \*\*Дополнительное задание: Реализация системного вызова dump2:\*\* Мы расширили функциональность операционной системы, добавив системный вызов 'dump2', который позволяет получить значения регистров заданного процесса. Этот вызов принимает три аргумента: номер процесса ('pid'), номер регистра ('register\_num'), ('return\_value'), :

- Если вызывающий процесс не имеет прав на чтение требуемого регистра, возвращали -1. - Если процесса с указанным 'pid' не существует, возвращали -2. - Если передан некорректный номер регистра, возвращали -3. - Если не удалось записать данные по переданному адресу, возвращали -4.

Мы использовали функцию 'copyout' для безопасной записи данных в user-space. Затем успешно запустили утилиту 'dump2tests', которая автоматически проверила функциональность нашего системного вызова 'dump2'.

В целом, выполненная работа дала нам обширный опыт в разработке и доработке операционной системы xv6, работе с системными вызовами, межпроцессным взаимодействием через pipes, а также работой с регистрами процессов. Это позволило нам более глубоко понять внутреннее устройство операционных систем и улучшить навыки программирования на С в контексте операционных систем.