Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**ПОЛЬЗОВАТЕЛЬСКИЕ ПРОГРАММЫ. АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ**

по дисциплине «Операционные системы»

Выполнил

студенты гр. 5131001/30002 Мишенев Н. С.

<*подпись*>

Руководитель

программист Огнёв Р. А.

<*подпись*>

Санкт-Петербург

2024г.

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 2](#_Toc1440144875)

[ХОД РАБОТЫ 3](#_Toc32228126)

[1. Внесённые модификации в исходный код ОС Pintos. 4](#_Toc632280762)

[> thread.h 4](#_Toc2135224285)

[> thread.c 4](#_Toc1304596462)

[> process.c 4](#_Toc1097593062)

[> process.h 4](#_Toc1972663151)

[> syscall.c 5](#_Toc346779105)

[2. Диаграмма состояний ожидания, передачи аргументов и результатов между основными функциями. 6](#_Toc898007534)

[3. Анализ тестовых программ и вывода тестов. 6](#_Toc1795947286)

[ВЫВОД 6](#_Toc1381343645)

# **ЦЕЛЬ РАБОТЫ**

Цель работы – изучение механизмов передачи параметров пользовательским программам и реализация такого механизма в архитектуре 80x86 с использованием стека.

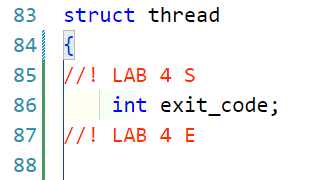
# **ХОД РАБОТЫ**

## **1. Внесённые модификации в исходный код ОС Pintos.**

В ходе выполнения лабораторной работы были внесены модификации в файлы: **thread.h, thread.c, process.c, process.h, syscall.c.** Рассмотрим изменения в этих файлах по порядку.

### **> thread.h**

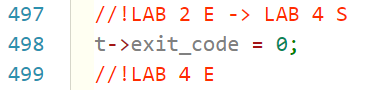
В данном файле, изменению подверглись поля структуры процесса. Было добавлено поле, которое содержит в себе код завершения процесса.



*Рис. 1. Новое поле* ***exit\_code*** *в структуре процесса.*

### **> thread.c**

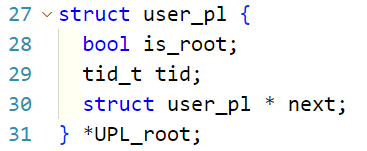
В этом файле, изменения были внесены в функцию инициализации процесса, для инициализации нового поля нулём.



*Рис. 2. Инициализация нового поля* ***exit\_code*** *в функции* ***init\_thread(...).***

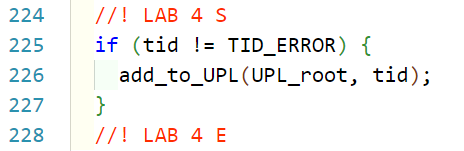
### **> process.c**

В данный файл было внесено два значительных изменения. Создан список пользовательских процессов (User Process List / UPL), он представлен односвязным списком, в котором хранится **tid** процесса, созданного функцией **process\_execute(...)**.



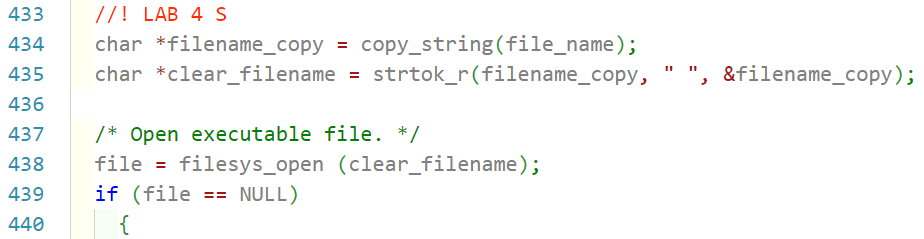
*Рис. 3. Реализация UPL.*

Добавление в этот список происходит, если процесс был успешно создан и получил свой **tid.**



*Рис. 4. Добавление процесса в UPL.*

Также, в нескольких местах, куда должно передаваться имя вызываемого пользовательского процесса, происходит отделение имени от аргументов с помощью одинаковой конструкции, в которой используется функция **strtok\_r(...).**

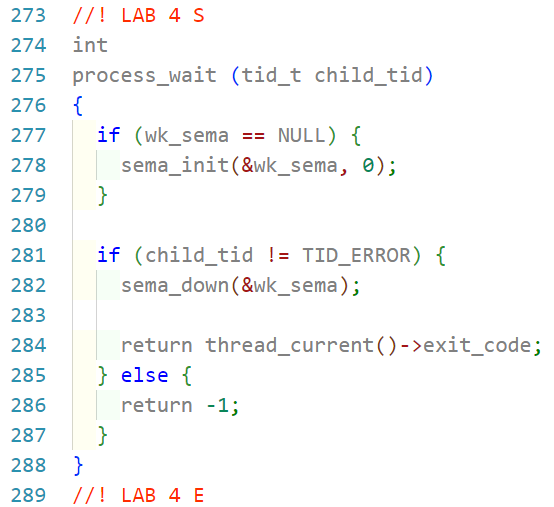


*Рис. 5. Отчистка имени от аргументов.*

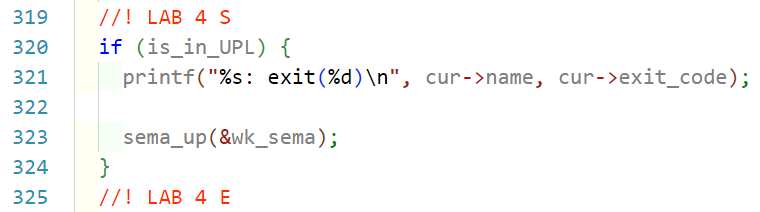
Имя будет получено при первой итерации по токену и помещено в переменную **clear\_filename.**

Далее, была модернизирована функция **load(...).** В нее добавлена функция **parse\_args\_to\_stack(...)** внутри которой происходит разбивка переданного имени процесса на аргументы и последующее перемещение их в стэк. (Приложение А).

Ожидание завершения дочернего процесса реализовано с помощью одного семафора, который инициализируется и опускается в функции **process\_wait()**, и поднимается в функции **process\_exit().**



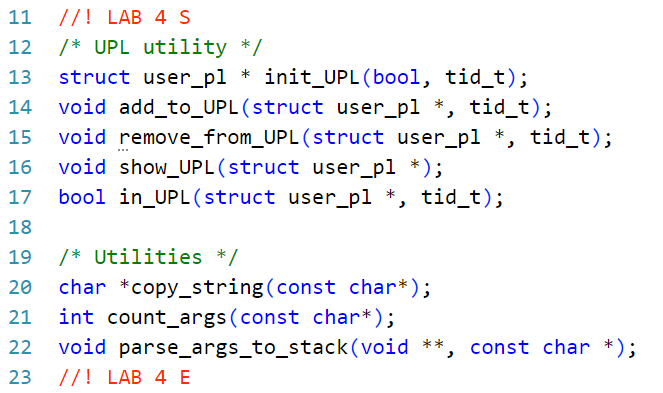
*Рис. 6. Модернизированная функция* ***process\_wait()***



*Рис. 7. Модернизированная функция* ***process\_exit()***

### **> process.h**

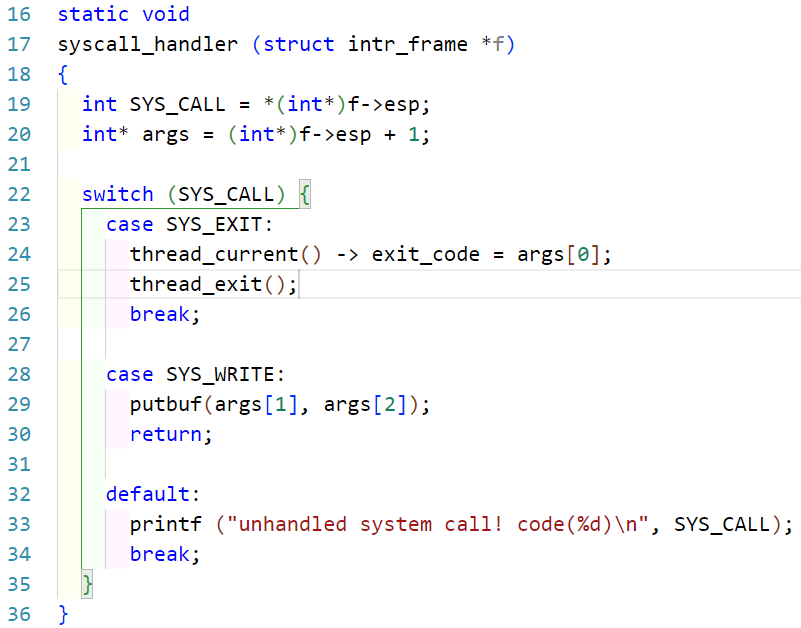
В соответствующий заголовочный файл были внесены прототипы созданных функций для **UPL** и для парсинга аргументов.



*Рис. 8. Прототипы реализованных функций в* ***process.c***

### **> syscall.c**

Также была реализована простейшая поддержка минимального набора системных вызовов **WRITE** и **EXIT** для пользовательских программ.

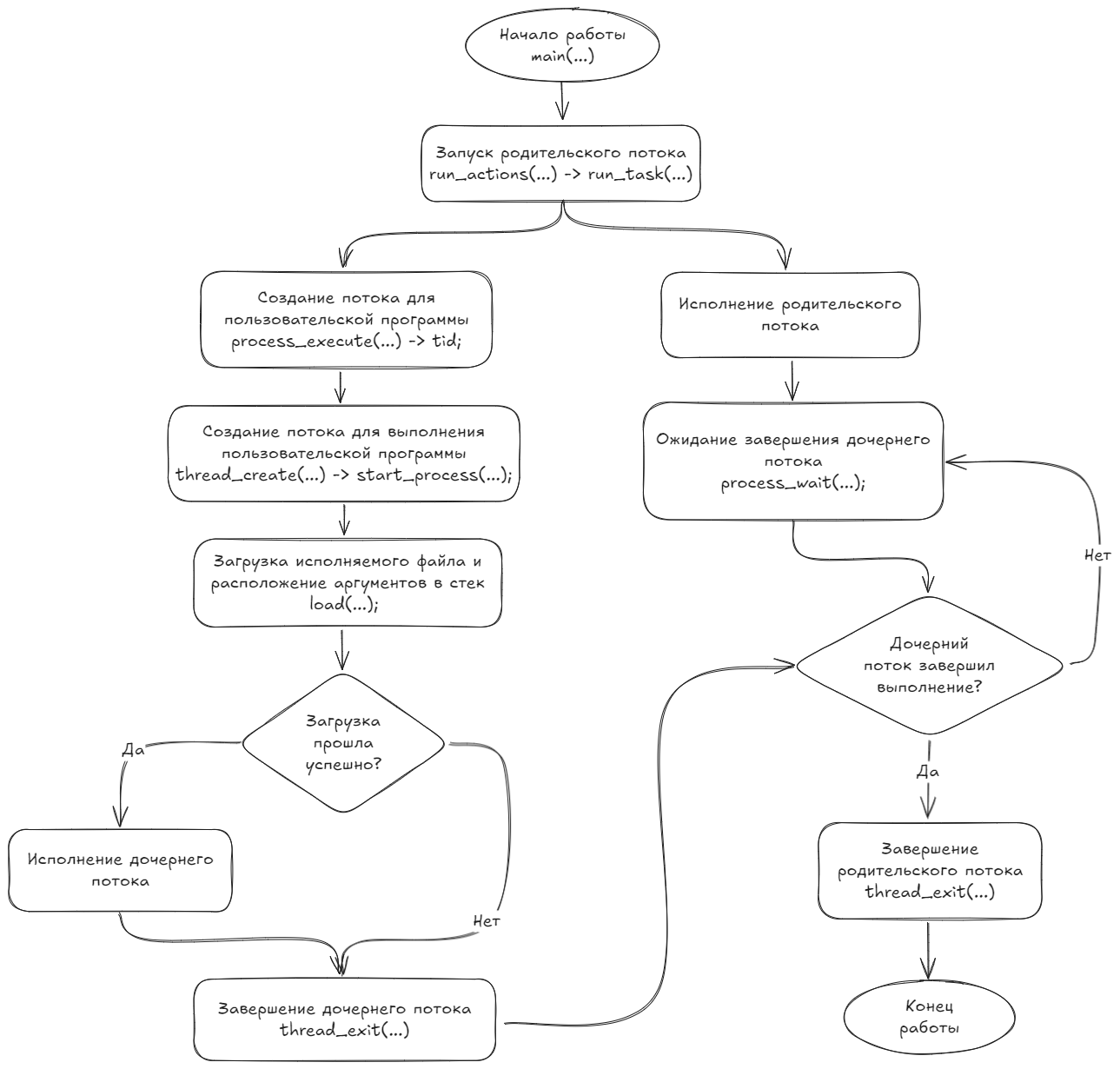


*Рис. 9. Реализация поддержки системных вызовов****.***

Реализации данных системных вызовов были взяты из методических рекомендаций и внесены в функцию **syscall\_handler(...).**

## **2. Диаграмма состояний ожидания, передачи аргументов и результатов между основными функциями.**

На основании модернизированного алгоритма была построена диаграмма состояний:



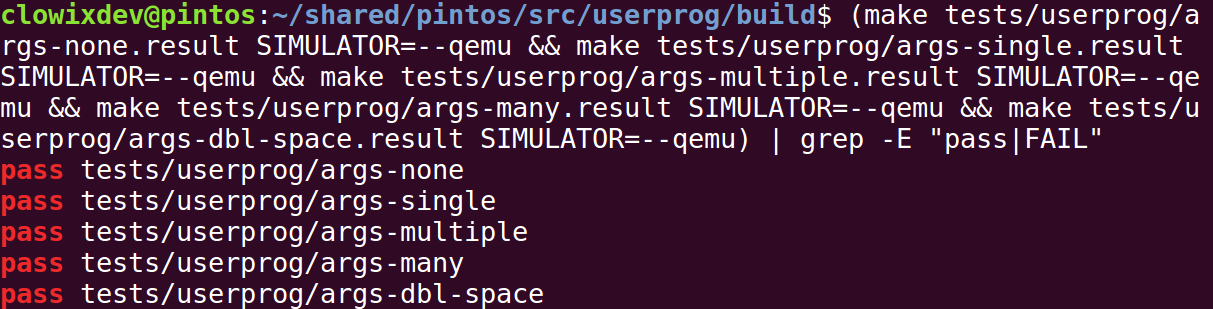
*Рис. 10. Диаграмма состояний****.***

## **3. Анализ тестовых программ и вывода тестов.**

Для проверки работоспособности решения было предоставлено 5 тестов: **args-none, args-single, args-multiple, args-many, args-dbl-space**. Рассмотрим каждый из них отдельно.

* args-none - в данном тесте, пользовательская программа вызывается без аргументов.
* args-single - в данном тесте, пользовательская программа вызывается с одним аргументом.
* args-nultiple - в данном тесте, пользовательская программа вызывается с несколькими аргументами, введёнными через пробел.
* args-many - в данном тесте, пользовательская программа вызывается с очень большим количеством аргументов, около 22.
* args-dbl-space - в данном тесте, пользовательская программа вызывается с несколькими аргументами, отделёнными двойным знаком пробела.

Разработанное решение успешно проходит все представленные тесты:



*Рис. 11. Успешное прохождение всех тестов.*

# **ВЫВОД**

В ходе работы были изучены механизмы передачи параметров пользовательским программам, а также реализован такой механизм в архитектуре 80x86 с использованием стека. Были изучены и имплементированы простейшие системные вызовы exit и write.

# **ПРИЛОЖЕНИЕ А**

Листинг 1 - функция parse\_args\_to\_stack(...)

/\* Function that will FILE\_NAME string and add all arguments to stack,

move ESP as it adds arguments to stack. \*/

void parse\_args\_to\_stack(void \*\*esp, const char \*file\_name) {

char \*arg\_p = (char \*)malloc(sizeof(char) \* (strlen(file\_name) + 1));

int argc = count\_args(file\_name);

char \*\*argv = (char \*\*)calloc(argc + 1, sizeof(char\*));

uint32\_t \*argv\_p = (uint32\_t\*)calloc(argc + 1, sizeof(uint32\_t));

char \*filename\_copy = copy\_string(file\_name);

int idx = 0;

while ((arg\_p = strtok\_r(filename\_copy, " ", &filename\_copy))) {

if (arg\_p != NULL) {

argv[idx] = arg\_p;

idx++;

}

}

for (int arg\_idx = argc - 1; arg\_idx >= 0; arg\_idx--) {

size\_t arg\_len = sizeof(char) \* strlen(argv[arg\_idx]) + 1;

\*esp -= arg\_len;

memcpy(\*esp, argv[arg\_idx], arg\_len);

argv\_p[arg\_idx] = (uint32\_t)\*esp;

}

uint32\_t align = (uint32\_t) \*esp % 4;

\*esp -= align;

memcpy(\*esp, &argv[argc], align);

for (int arg\_idx = argc; arg\_idx >= 0; arg\_idx--) {

\*esp -= sizeof(uint32\_t);

memcpy(\*esp, &argv\_p[arg\_idx], sizeof(uint32\_t\*));

}

uint32\_t argv\_ptr = \*esp;

\*esp -= sizeof(uint32\_t);

memcpy(\*esp, &argv\_ptr, sizeof(uint32\_t\*\*));

\*esp -= (sizeof(int));

memcpy(\*esp, &argc, sizeof(int));

\*esp -= (sizeof(void\*));

memcpy(\*esp, &argv[argc], sizeof(void\*));

free(argv);

free(argv\_p);

free(arg\_p);

return;

}