1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Высшая школа кибербезопасности»**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «ПОЛЬЗОВАТЕЛЬСКИЕ ПРОГРАММЫ. АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/30003 Шевчук Н.Е.

<*подпись*>

Преподаватель Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2024

**Оглавление**

[1 Цель работы 3](#_Toc184232071)

[2 Ход работы 3](#_Toc184232072)

[2.1 Функция load() 3](#_Toc184232073)

[2.2 Функция process\_wait() 4](#_Toc184232074)

[2.3 Функция process\_exit() 4](#_Toc184232075)

[2.4 Функция syncall\_handler() 5](#_Toc184232076)

[2.5 Диаграмма состояний 5](#_Toc184232077)

[2.6 Анализ тестовых программ 6](#_Toc184232078)

[3 Вывод 8](#_Toc184232079)

# Цель работы

Изучить механизмы передачи параметров пользовательским программам и реализовать такой механизма в архитектуре 80x86 с использованием стека.

# Ход работы

Для корректной обработки вызова пользовательских программ в командоной строке в исходные коды были внесены следующие изменения:

## Функция load()

В начале функции load() каталога procces.c было добавлено извлечение имени файла, для его корректного открытия с помощью функции strtok\_r. Это отделённое имя записывается как имя потока, для корректноговывода в process\_exit().

char\* token;

char\* name = strtok\_r(file\_name, " ", &token);

strlcpy(thread\_current()->name, name, strlen(name) + 1);

Далее, после инициализации стека и установки флага saccess помещаем элементы командной строки в стек в обратном порядке. Сначала копируем имя, отделённое выше, в стек и сохраняем указатель на него в массиве argv:

char\* argv[100];

int argc = 0;

size\_t length;

length = strlen(name) + 1;

\*esp -= length;

strlcpy(\*esp, name, length);

argv[argc++] = \*esp;

Затем, копируем в стек значения остальных аргументов входной командной строки и также сохраняем указатели на них в массиве argv:

char\* save\_ptr;

char\* token2;

for(token2 = strtok\_r(token, " ", &save\_ptr); token2 != NULL; token2 = strtok\_r(NULL, " ", &save\_ptr)) {

length = strlen(token2) + 1;

\*esp -= length;

strlcpy(\*esp, token2, length);

argv[argc++] = \*esp;

}

Затем, сначала вне цикла записываем в стек последний элемент массива – нулевой указатель. А затем, все остальные указатели из массива argv в обратном порядке:

\*esp -= 4;

\*((void\*\*)\*esp)=NULL;

int k;

for (k = argc - 1; k >= 0; k--) {

\*esp -= 4;

\*((void\*\*)\*esp)=argv[k];

}

Затем, в стек последовательно помещаются указатель на сам массив argv, количество элементов args в массиве argv и мнимый адрес возврата.

void \*argv\_ptr = \*esp;

\*esp -= 4;

\*((void\*\*)\*esp)=argv\_ptr;

\*esp -= 4;

\*((int\*)\*esp)=argc;

\*esp -= 4;

\*((void\*\*)\*esp)=NULL;

## Функция process\_wait()

Для того, чтобы родительский поток дожидался завершения дочернего потока был добавлен семафор sema. Инициализируется он в process\_execute() до создания дочернего потока. Блокируется в process\_wait() и поднимается в process\_exit(), когда дочерний поток завершает свою работу.

int

process\_wait (tid\_t child\_tid UNUSED)

{

**sema\_down(&sema);**

return -1;

}

## Функция process\_exit()

Как сказано выше, в функцию process\_exit() помещен подъем семафора для сигнализирования родительскому потоку о завершении дочернего. Также добавлен вывод диагностического сообщения. В структуру thread в каталоге thread.h добавлено новое поле exit\_status.

void

process\_exit (void)

{

struct thread \*cur = thread\_current ();

uint32\_t \*pd;

/\* Destroy the current process's page directory and switch back

to the kernel-only page directory. \*/

pd = cur->pagedir;

if (pd != NULL)

{

/\* Correct ordering here is crucial. We must set

cur->pagedir to NULL before switching page directories,

so that a timer interrupt can't switch back to the

process page directory. We must activate the base page

directory before destroying the process's page

directory, or our Sactive page directory will be one

that's been freed (and cleared). \*/

cur->pagedir = NULL;

pagedir\_activate (NULL);

pagedir\_destroy (pd);

**printf("%s: exit(%d)\n", cur->name, cur->exit\_status);**

**sema\_up(&sema);**

}

}

## Функция syncall\_handler()

В каталоге syscall.c была добавлена минимальная реализация системных вызовов exit и call. При реализации вызова exit код выxода запоминается в поле exit\_status потока.

static void

syscall\_handler (struct intr\_frame \*f)

{

if (\*(int\*)f->esp == SYS\_WRITE) {

putbuf(((const char\*\*)f->esp)[2], ((size\_t\*)f->esp)[3]);

return;

}

else if (\*(int\*)f->esp == SYS\_EXIT) {

int exit\_status = ((size\_t\*)f->esp)[1];

thread\_current()->exit\_status = exit\_status;

thread\_exit();

return;

}

else {

printf("system call!\n");

thread\_exit();

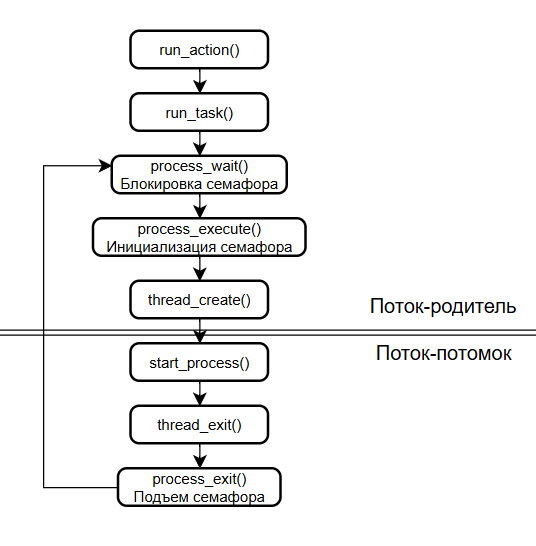
return;

}

}

## Диаграмма состояний

На рисунке ниже представлена диаграмма состояний ожидания, передачи аргументов и результатов выполнения между основными функциями:



*Рисунок 1. Диаграмма состояний и связи функций*

## Анализ тестовых программ

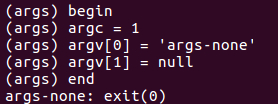
После внесённых изменений представленные тесты выполняются корректно:



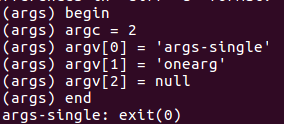
*Рисунок 2. Результат работы тестов*

Рассмотрим каждый тест подробнее:

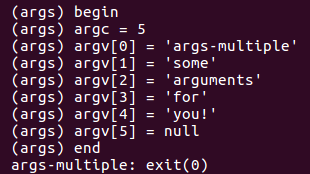
* args-none – запуск программы без аргументов;



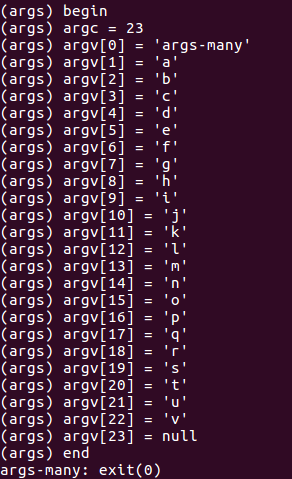
* args-single – запуск программы с одним аргументом;



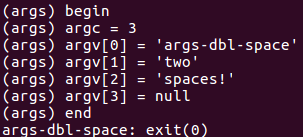
* args-multiple – запуск программы с небольшим количеством аргументов;



* args-many – запуск программы с большим количеством аргументов;



* args-dbl-space – запуск программы с аргументами, разделёнными двумя пробелами.



# Вывод

В ходе лабораторной работы были изучены механизмы передачи параметров пользовательским программам и реализован такой механизм в архитектуре 80x86 с использованием стека.