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

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.

```
83  struct thread
84  {
85  //! LAB 4 S
86      int exit_code;
87  //! LAB 4 E
88
```

*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.

```
497   //!LAB 2 E -> LAB 4 S
498   t->exit_code = 0;
499   //!LAB 4 E
```

*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(...)**.

```
27 struct user_pl {
28   bool is_root;
29   tid_t tid;
30   struct user_pl * next;
31 } *UPL_root;
```

Рис. 3. Реализация UPL.

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

```
224   //! LAB 4 S
225   if (tid != TID_ERROR) {
226     add_to_UPL(UPL_root, tid);
227   }
228   //! LAB 4 E
```

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(...).**

```
433   //! LAB 4 S
434   char *filename_copy = copy_string(file_name);
435   char *clear_filename = strtok_r(filename_copy, " ", &filename_copy);
436
437   /* Open executable file. */
438   file = filesys_open (clear_filename);
439   if (file == NULL)
440     {
```

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().**

```
273  //! LAB 4 S
274  int
275  process_wait (tid_t child_tid)
276  {
277    if (wk_sema == NULL) {
278      sema_init(&wk_sema, 0);
279    }
280
281    if (child_tid != TID_ERROR) {
282      sema_down(&wk_sema);
283
284      return thread_current()->exit_code;
285    } else {
286      return -1;
287    }
288  }
289  //! LAB 4 E
```

*Рис. 6. Модернизированная функция **process_wait()***

```
319  //! LAB 4 S
320  if (is_in_UPL) {
321    printf("%s: exit(%d)\n", cur->name, cur->exit_code);
322
323    sema_up(&wk_sema);
324  }
325  //! LAB 4 E
```

*Рис. 7. Модернизированная функция **process_exit()***

> process.h

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

```
11  //! LAB 4 S
12  /* UPL utility */
13  struct user_pl * init_UPL(bool, tid_t);
14  void add_to_UPL(struct user_pl *, tid_t);
15  void remove_from_UPL(struct user_pl *, tid_t);
16  void show_UPL(struct user_pl *);
17  bool in_UPL(struct user_pl *, tid_t);
18
19  /* Utilities */
20  char *copy_string(const char*);
21  int count_args(const char*);
22  void parse_args_to_stack(void **, const char *);
23  //! LAB 4 E
```

*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.

```
16  static void
17  syscall_handler (struct intr_frame *f)
18  {
19    int SYS_CALL = *(int*)f->esp;
20    int* args = (int*)f->esp + 1;
21
22    switch (SYS_CALL) {
23      case SYS_EXIT:
24        thread_current() -> exit_code = args[0];
25        thread_exit();
26        break;
27
28      case SYS_WRITE:
29        putbuf(args[1], args[2]);
30        return;
31
32      default:
33        printf ("unhandled system call! code(%d)\n", SYS_CALL);
34        break;
35    }
36  }
```

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:

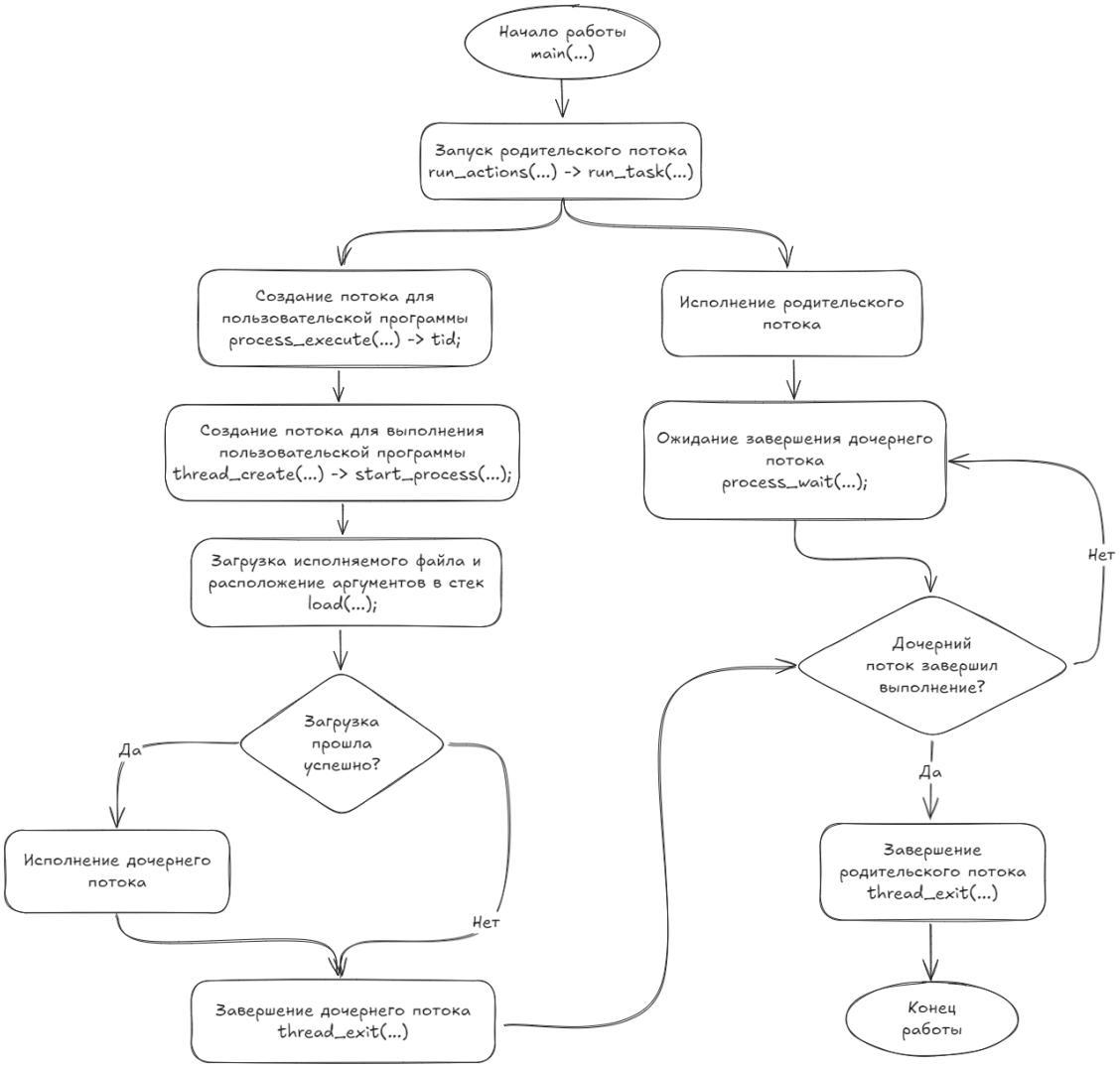

Рис. 10. Диаграмма состояний.

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:

```
clowixdev@pintos:~/shared/pintos/src/userprog/build$ (make tests/userprog/a
rgs-none.result SIMULATOR=--qemu && make tests/userprog/args-single.result
SIMULATOR=--qemu && make tests/userprog/args-multiple.result SIMULATOR=--qe
mu && make tests/userprog/args-many.result SIMULATOR=--qemu && make tests/u
serprog/args-dbl-space.result SIMULATOR=--qemu) | grep -E "pass|FAIL"
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
```

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;
}
```