Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА №4

«ПОЛЬЗОВАТЕЛЬСКИЕ ПРОГРАММЫ. АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ»

по дисциплине «Операционные системы»

Выполнил
студент гр. 4851003/10002 <подпись> Галкин К. К.

Руководитель
к. т. н. <подпись> Крундышев В.М.

Санкт-Петербург
2022

1. ЦЕЛЬ РАБОТЫ

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

2. ХОД РАБОТЫ

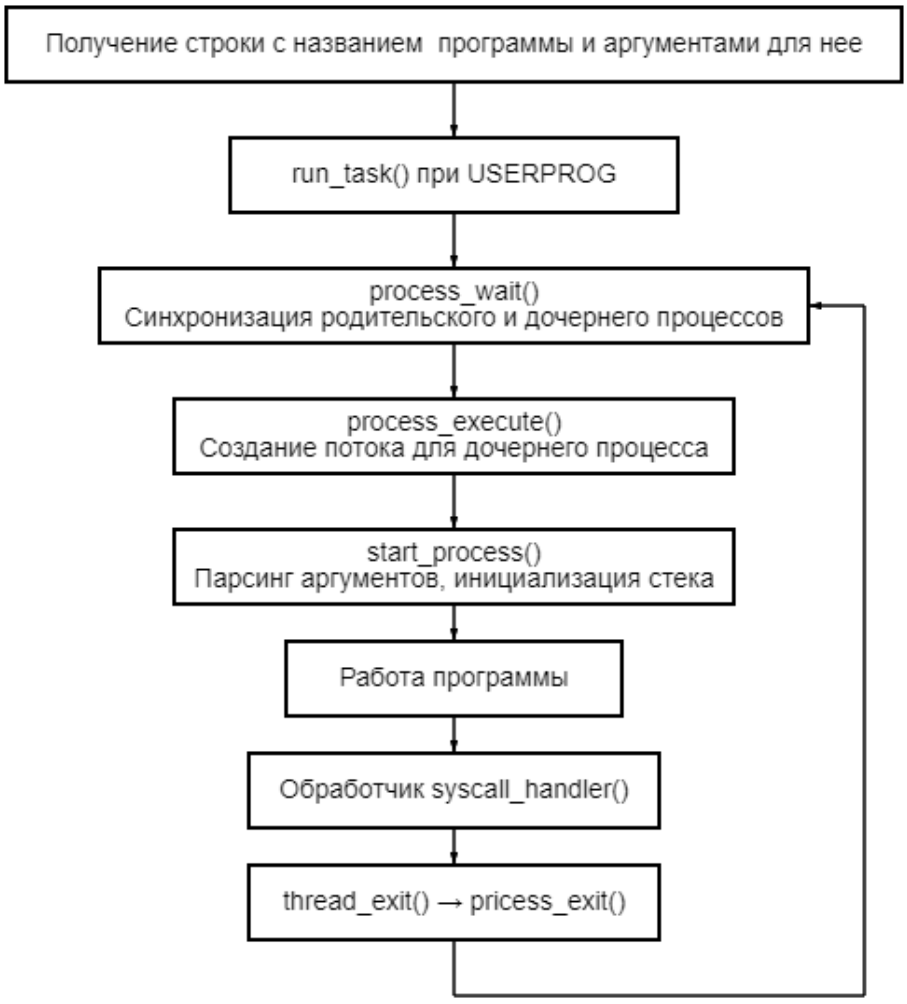
Основная задача – правильный парсинг аргументов командной строки при выполнении пользовательской программы. Пользовательская программа подгружается на виртуальный диск как исполняемый файл. Рассмотрим основные изменения в коде для решения задачи работы пользовательских программ.

- Thread.h:
  - Добавления поля error_code для структуры потока
- Thread.c:
  - Исправление ошибки с присваиванием имени к потоку. Были лишние символы в каждом названии, которое имеет пробел
- Syscall.c:
  - Добавление двух действий обработчика: вывод сообщения на экран (SYS_WRITE) или завершение пользовательского процесса (SYS_EXIT).
- Process.c
  - Добавлен вывод об окончании работы процесса в функции process_exit()
  - В функции load() реализован парсинг аргументов командной строки через strtok_r. Каждый аргумент записывается в стек по определенному адресу. После эти адреса запоминаются в отдельных массив args_pointers. Ограничение на количество аргументов командной строки – 25 – введено самостоятельно. Таким образом в стек записываются данные в следующем

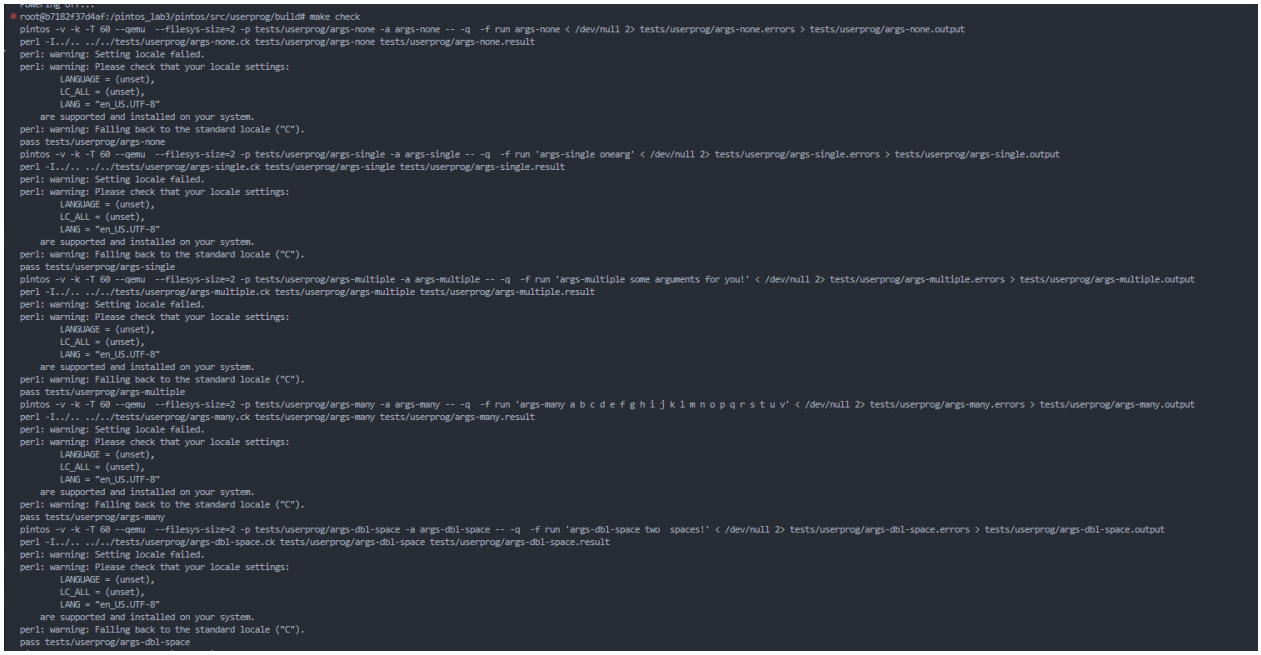
порядке: строки – аргументы командной строки, разделительные нулевые байты, адреса на аргументы командной строки, количество аргументов argc, нулевые байты. Такое расположение данных позволяет обеспечивать выравнивание данных и защиту от перетирания данных.

- Добавления семафора process_sema для контроля доступа к ресурсам. То есть необходимо, чтобы пользовательская программа заполучила ресурсы после инициализации, а не после того, как завершится main.

Диаграмма состояний ожидания, передачи аргументов и результатов выполнения между основными функциями:



3. РЕЗУЛЬТАТЫ РАБОТЫ



4. ВЫВОД

В ходе лабораторной работы был изучен принцип взаимодействия ОС Pintos с пользовательскими программами: как программы попадают в память системы, как выделяется под программы стек, как происходит работа с памятью, для чего и как используются аргументы командной строки пользовательской программы, как аргументы помещаются в выделенную память. Так же был изучен принцип выстраивания данных в стеке ОС Pintos и что такое системные вызовы.

ПРИЛОЖЕНИЕ

- Thread.h

```
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes). */
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Priority. */
    struct list_elem allelem;           /* List element for all threads list. */
    int error_code;                     /* Thread error code */
    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /* List element. */

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */
  };
```

- Thread.c

```
static void
init_thread (struct thread *t, const char *name, int priority)
{
  ASSERT (t != NULL);
  ASSERT (PRI_MIN <= priority && priority <= PRI_MAX);
  ASSERT (name != NULL);
  char *normal_name = (char *)name;
  for(int i = 0; i < strlen(normal_name); i++) {
    if(normal_name[i] == 0x20) {
      normal_name[i] = '\0';
      break;
  } …
```

- Syscall.c

```
static void

syscall_handler(struct intr_frame *f UNUSED)

{

  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_exit(); // вызов exit() должен завершать программу
  }
  else
  {
    printf("system call!\n");
    thread_exit();
  }
}
```

- Process.c

```
struct semaphore process_sema;
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. */
  printf("%s: exit(%d)\n", cur->name, cur->error_code);
…
tid_t process_execute(const char *file_name)
{
  char *fn_copy;
  tid_t tid;
  sema_init(&process_sema, 0);
…
int process_wait(tid_t child_tid UNUSED)
{
  sema_down(&process_sema);
  return -1;
}
void process_exit(void)
{
…
```

```
sema_up(&process_sema);
}
bool load(const char *file_name, void (**eip)(void), void **esp)
{
…
char *token, *saveptr;
char *argv[25];
int argc = 0;
uint32_t *args_pointers[25];
…
for (token = strtok_r((char *)file_name, " ", &saveptr); token != NULL; token = strtok_r(NULL, "
", &saveptr))
  {
    argv[argc] = token;
    argc++;
  }
  /* Open executable file. */
  file = filesys_open(argv[0]);
…
for (int j = argc - 1; j >= 0; j--)
  {
    *esp = *esp - sizeof(char) * (strlen(argv[j]) + 1);
    memcpy(*esp, argv[j], sizeof(char) * (strlen(argv[j]) + 1));
    args_pointers[j] = (uint32_t *)*esp;
  }
  /* Allocate space for & add the null sentinel. */
  *esp = *esp - 4;
  (*(int *)(*esp)) = 0;

  *esp = *esp - 4;
  for (int k = argc - 1; k >= 0; k--)
  {
    (*(uint32_t **)(*esp)) = args_pointers[k];
    *esp = *esp - 4;
  }
  /* Push onto the stack a pointer to the pointer of the address of the
```

```
   first argument in the list of arguments. */
  (*(uintptr_t **)(*esp)) = *esp + 4;

  /* Push onto the stack the number of program arguments. */
  *esp = *esp - 4;
  memcpy(*esp, &argc, sizeof(int));

  /* Push onto the stack a fake return address, which completes stack initialization. */
  *esp = *esp - 4;
  (*(int *)(*esp)) = 0;

  success = true;
…
}
```