Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Демина В.А.

Группа: М8О–208Б–21

Вариант: 22

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

**Задание**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи

один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут

возникнуть в результате работы.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

**Общие сведения о программе**

Программа компилируется из файла main.c. Отличительной особенностью программы (относительно кода лабораторной) является использование системного вызова mmap. Этот системный вызов запускается с ключом MAP\\_ANONYMOUS, это означет, что файл, отображаемый в память, будет виртуальным. Также в этой лабораторной используется обработчик системного сигнала (сигнала о завершении работы родительского процесса).

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap.
2. Изучить принципы работы системных сигналов.
3. Изучить принципы работы системных сигналов.
4. Изучить принципы работы системных сигналов.
5. Изучить принципы работы системных сигналов.

**Основные файлы программы**

**main.c:**

#include <fcntl.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <unistd.h>

int is\_it\_time\_to\_terminate = 0;

const int OFFSET = sizeof(int);

void update\_is\_it\_time\_to\_terminate(int num) {

  is\_it\_time\_to\_terminate = 1;

}

void reverse(char name[])

{

    char tmp;

    int len = strlen(name);

  --len;

    for (int i = 0; i < len; i++) {

        tmp = name[len - 1];

        name[len - 1] = name[i];

        name[i] = tmp;

        len--;

    }

}

void child\_work(char\* from, int to) {

    int idx = OFFSET;

    char ch;

    char string[100];

    int i = 0;

  while (1) {

    if (idx < OFFSET + ((int\*)from)[0]) {

      string[i++] = from[idx];

      ch = from[idx];

      string[i] = '\0';

      if ((ch == '\n')||(ch == '\r')){

        reverse(string);

    write(to, string, i);

    i = 0;

      }

      ++idx;

    } else {

      if (is\_it\_time\_to\_terminate) {

        break;

      }

    }

  }

  close(to);

}

void parrent\_work(pid\_t child1, char\* child\_map1, pid\_t child2, char\* child\_map2) {

  char ch;

  int idx1 = OFFSET;

  int idx2 = OFFSET;

  srand(time(NULL));

  int n = (rand() % 10) + 1;

  while (read(STDIN\_FILENO, &ch, 1) > 0) {

    if (n > 2) {

      child\_map1[idx1++] = ch;

    } else {

      child\_map2[idx2++] = ch;

    }

    if (ch == '\n') {

      if (n > 2) {

        ((int\*)child\_map1)[0] = idx1 - OFFSET;

      } else {

        ((int\*)child\_map2)[0] = idx2 - OFFSET;

      }

      n = (rand() % 10) + 1;

    }

  }

  kill(child1, SIGUSR1);

  kill(child2, SIGUSR1); //сигнал о зовершении работы родительского процесса

  int res1;

  int res2;

  waitpid(child1, &res1, 0); // ждем завершения дочерних процессов

  waitpid(child2, &res2, 0);

  if (res1 != 0 || res2 != 0) {

    fprintf(stderr, "Something ended ne tak!\n%d %d\n", res1, res2);

  }

}

int read\_name\_and\_open\_file() {

  const size\_t FILE\_NAME\_SIZE = 64;

  char f\_name[FILE\_NAME\_SIZE];

  char buf[1];

  int idx = 0;

  while (idx < FILE\_NAME\_SIZE && read(STDIN\_FILENO, buf, 1) > 0) {

    if (buf[0] == '\n') {

      break;

    }

    f\_name[idx++] = buf[0];

  }

  f\_name[idx++] = '\0';

  return open(f\_name, O\_WRONLY | O\_TRUNC);

}

void error(char\* buf, size\_t size) { write(STDERR\_FILENO, buf, size); }

void check\_file\_id(int id) {

  if (id == -1) {

    error("File not found\n", 15);

    exit(-1);

  }

}

void\* check\_map\_creation() {

  void\* m\_file = mmap(NULL, 2048, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

  //  mmap  ( void \* address , size\_t  length , int  protect , int  flags , int  filedes(дискриптор) ,off\_t  offset(смещение))

  // 2048 байт нужно отобразить в память

  // PROT\_READ | PROT\_WRITE - права на чтение и запись

  // MAP\_SHARED  - флаг для совместного использования отображения со всеми другими процессами, которые отображаются на этот объект

  // MAP\_ANONYMOUS - флаг для создания анонимного сопоставления (файл отображаемый в память будет виртуальным)

  if (m\_file == MAP\_FAILED) { // сбой (иначе 0)

    error("Cannot create mmap\n", 19);

    exit(-2);ы

  }

  ((int\*)m\_file)[0] = 0;

  return m\_file;

}

int check\_fork() {

  int fd = fork();

  if (fd == -1) {

    error("Cannot create process\n", 22);

    exit(-3);

  }

  return fd;

}

void add\_signals() {

  void (\*func)(int);

  func = signal(SIGUSR1, update\_is\_it\_time\_to\_terminate);

  // SIGUSR1 - сигнал отданный на распоряжение пользователю

  if (func == SIG\_IGN) { // сигнал игнорируется

    error("Cannot add signal\n", 18);

    exit(-4);

  }

}

int main(int argc, char\* argv[]) {

  add\_signals();

  int f1 = read\_name\_and\_open\_file();

  check\_file\_id(f1);

  int f2 = read\_name\_and\_open\_file();

  check\_file\_id(f2);

  char\* m\_file1 = check\_map\_creation();

  pid\_t child1 = check\_fork();

  if (child1 == 0) {

    close(f2);

    child\_work(m\_file1, f1);

    return 0;

  }

  close(f1); // закрывает mmap для f1

  char\* m\_file2 = check\_map\_creation();

  pid\_t child2 = check\_fork();

  if (child2 == 0) {

    child\_work(m\_file2, f2);

    return 0;

  }

  close(f2); // закрывает mmap для f2

  parrent\_work(child1, m\_file1, child2, m\_file2);

  return 0;

}

**Вывод**

В процессе работы над лабораторной я научилась основам работы с файлами отображаемыми в память и системными сигналами в Си. Файлы отображаемые в память являются крайне полезным инструментом, так как в ряде случаем эта технология способна значительно повысить эффективность работы программы. Системные сигналы оказались полезными при работе над последующими лабораторными.