Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Студент: Велиев Рауф Рамиз оглы
Группа: М8О-209Б-23
Вариант: 4
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/velievrauf/OS/tree/main/lab_3

Постановка задачи

Цель работы

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.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление parent child(args: fileName) pipe1 pipe2 In/out User File In Out первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

- Создание разделяемой памяти:

Родительский процесс создает разделяемую память, доступную для чтения и записи как ему, так и дочернему процессу.

- Форк процесса:

Родительский процесс создает дочерний процесс с помощью fork().

- Передача данных:

Родительский процесс считывает вводимые числа, сохраняет их в память и уведомляет дочерний процесс, установив соответствующий флаг.

- Обработка данных дочерним процессом:

Дочерний процесс выполняет деление первого числа на последующие. Если встречается деление на ноль, он записывает сообщение об ошибке в разделяемую память и завершает работу.

- Возвращение результата:

Если деление успешно, дочерний процесс сохраняет результат в разделяемую память и уведомляет родительский процесс.

- Вывод результата:

Родительский процесс считывает результат из разделяемой памяти, выводит его пользователю, освобождает ресурсы и завершает выполнение.

Исходный код

lab3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/wait.h>

#define SHARED_MEMORY_NAME "/my_shm"
#define SHARED_MEMORY_SIZE sizeof(struct SharedMemory)

struct SharedMemory {
    float numbers[100];
```

```
    int count;
    volatile int flag;
    char result[256];
};

int main() {
    int shm_fd = shm_open(SHARED_MEMORY_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, SHARED_MEMORY_SIZE) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    struct SharedMemory *shm_ptr = mmap(NULL, SHARED_MEMORY_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        close(shm_fd);

        execl("./kid", "kid", NULL);
        perror("execl");
        exit(EXIT_FAILURE);
    } else {
        close(shm_fd);

        printf("Введите числа (разделенные пробелами), завершите <endline>: ");
        float numbers[100];
        int count = 0;

        while (scanf("%f", &numbers[count]) != EOF) {
            count++;
        }

        memcpy(shm_ptr->numbers, numbers, sizeof(float) * count);
        shm_ptr->count = count;
        shm_ptr->flag = 1;

        while (shm_ptr->flag != 2) {
            usleep(100000);
        }

        printf("Результат: %s\n", shm_ptr->result);

        munmap(shm_ptr, SHARED_MEMORY_SIZE);
        shm_unlink(SHARED_MEMORY_NAME);
```

```
    wait(NULL);
  }

  return 0;
}
```

kid.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#define SHARED_MEMORY_NAME "/my_shm"
#define SHARED_MEMORY_SIZE sizeof(struct SharedMemory)

struct SharedMemory {
  float numbers[100];
  int count;
  volatile int flag;
  char result[256];
};

int main() {
  int shm_fd = shm_open(SHARED_MEMORY_NAME, O_RDWR, 0666);
  if (shm_fd == -1) {
    perror("shm_open");
    exit(EXIT_FAILURE);
  }

  struct SharedMemory *shm_ptr = mmap(NULL, SHARED_MEMORY_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
  if (shm_ptr == MAP_FAILED) {
    perror("mmap");
    exit(EXIT_FAILURE);
  }

  while (1) {
    if (shm_ptr->flag == 1) {
      float result = shm_ptr->numbers[0];
      for (int i = 1; i < shm_ptr->count; i++) {
        if (shm_ptr->numbers[i] == 0) {
          snprintf(shm_ptr->result, sizeof(shm_ptr->result), "Ошибка: деление на ноль");
          shm_ptr->flag = 2;
          return 0;
        }
        result /= shm_ptr->numbers[i];
      }

      snprintf(shm_ptr->result, sizeof(shm_ptr->result), "Результат: %.2f", result);
      shm_ptr->flag = 2;
    }
    usleep(100000);
  }

  munmap(shm_ptr, SHARED_MEMORY_SIZE);
  close(shm_fd);
```

```
    return 0;
}
```

Демонстрация работы программы

./lab3

Введите числа (разделенные пробелами), завершите <endline>: 120 4 2

Результат: 15.00

./lab3

Введите числа (разделенные пробелами), завершите <endline>: -15 0

Результат: Ошибка: деление на ноль

Выводы

Memory-mapped files — это технология, позволяющая связать содержимое файла с адресным пространством процесса. Благодаря этому файлы можно обрабатывать так же, как оперативную память, без необходимости вручную читать или записывать данные. Такой способ значительно упрощает и ускоряет работу, особенно при обработке больших файлов. Этот механизм особенно полезен для совместного доступа нескольких процессов к одному файлу и создания высокоэффективных кешей. Memory-mapped files представляют собой удобный и производительный инструмент для работы с данными.

В рамках лабораторной работы был продемонстрирован практический пример применения memory-mapped files для организации обмена данными между процессами. Этот подход позволил эффективно передать числа из родительского процесса в дочерний, выполнить вычисления и вернуть результат.