Министерство науки и высшего образования

Московский Авиационный Институт
(национальный исследовательский университет)

ЛАБОРАТОРНАЯ РАБОТА №4
по курсе операционные системы I семестр 2022/2023

Студент: Катин Иван Вячеславович
Группа: М8О-210Б-21
Вариант: 9
Преподователь : Миронов Евгений Сергеевич
Оценка:
Дата:
Подпись:

Москва, 2022

Содержание

Постановка задачи

Вариант 9.
В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.

Цель раобты

Приобритение практических навыков в:

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

Алгоритм решения

Открываем переданный файл на чтение. Создаем несколько отображений файлов в память(для этого используем функции **shm_open** и **mmap**): один для передачи результатов, второй для синхронизации чтения и записи результатов(дочерний процесс не может записать результат вычеслений, пока родительский процесс не вывел прошлый в стандартный поток вывода).
Создаем дочерний процесс, для дочернего процесса заменяем стандартный поток входа переданным файлом(с помощью **dup2**). Запускаем программу для дочернего процесса. Получаем указатели на отображаемые участки памяти. Читаем строку делим и проверяем деление на 0. Если родительский процесс не считал предыдущий результат, ждем, иначе записываем результат в участок памяти(*pointer_ data_ exchange*) и записываем в участок памяти *is_ child* «0»(«0» значит, что новый результат записан, можно читать, нельзя писать, «1» можно производить запись)
Для родительского процесса. Ждем, когда значение в участке памяти *is_ child* будет равняться «0», далее выводим результат в стандартный поток вывода и записываем в участк памяти *is_ child* «1», т.е. дочерний процесс может производить запись.

Код программы

Родительский процесс

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

#include "constants.h"

void get_file_name(char *buffer) {
    for (int index = 0; index < strlen(buffer); ++index) {
        if (buffer[index] == '\n') {
            buffer[index] = '\0';
            return;
        }
    }
}

int main() {
    shm_unlink(file_name);
    shm_unlink(is_child_name);

    int is_child_id = shm_open(is_child_name, O_CREAT | O_RDWR, 0666);
    handle_error(is_child_id == -1,
                 "shm_open error");
    handle_error(ftruncate(is_child_id, size) == -1, "truncate error");
    char* is_child = (char*) mmap(NULL, size,PROT_READ | PROT_WRITE, MAP_SHARED, is_child_id, 0);

    strcpy(is_child, "1");

    int file_descriptor_exchange = shm_open(file_name, O_CREAT | O_RDWR, 0666);
    handle_error(ftruncate(file_descriptor_exchange, size) == -1, "truncate error");
    handle_error(file_descriptor_exchange == -1,
                 "shm_open error");

    void* result_data_exchange = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED,
        file_descriptor_exchange, 0);
    handle_error(result_data_exchange == MAP_FAILED, "nmap error");
    strcpy(result_data_exchange, "\0");


    char buffer[50];
    handle_error(read(fileno(stdin), buffer, sizeof(buffer)) <= 0,
                 "error reading form stdin");

    get_file_name(buffer);

    int file_descriptor = open(buffer, O_RDONLY);
    handle_error(file_descriptor == -1, "Can't open file");

    pid_t process_id = fork();
    handle_error(process_id < 0, "process creation error");

    if (process_id == 0) {
        handle_error(dup2(file_descriptor, STDIN_FILENO) < 0, "error dub");
        handle_error(execl("child", file_name, NULL) < 0, "error process");
```

```
56
57      } else {
58          while(strcmp(is_child, "1") == 0);
59          char* buffer_for_float = "start";
60          while (strcmp(buffer_for_float, "\0" ) != 0){
61              while(strcmp(is_child, "1") == 0);
62              buffer_for_float = (char*) result_data_exchange;
63              printf("%s\n", buffer_for_float);
64              strcpy(is_child, "1");
65
66          }
67      }
68      shm_unlink(file_name);
69      shm_unlink(is_child_name);
70      return 0;
71  }
```

Дочерний процесс

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

#include "constants.h"

int main(int argc, char* argv[]){
    int is_child_id = shm_open(is_child_name, O_CREAT | O_RDWR, 0644);
    handle_error(is_child_id == -1,
                "shm_open error");
    handle_error(ftruncate(is_child_id, size) == -1, "truncate error");
    char* is_child = (char*) mmap(NULL, size,PROT_READ | PROT_WRITE, MAP_SHARED, is_child_id, 0);

    int file_descriptor_exchange = shm_open(file_name, O_CREAT | O_RDWR, mods);
    handle_error(ftruncate(file_descriptor_exchange, size) == -1, "truncate error");
    handle_error(file_descriptor_exchange == -1,
                "shm_open error");

    void* pointer_data_exchange = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED,
        file_descriptor_exchange, 0);

    char character;
    float result;
    bool is_first_number = true;

    while ((read(fileno(stdin), &character, 1)) >0){
        char* buffer = malloc(sizeof(char) * 50);
        int index = 0;
        while(character != ' ' && character != '\n' && character != '\0'){
            buffer[index++] = character;
            if(read(fileno(stdin), &character, 1) <= 0) {
                character = EOF;
                break;
            }
        }

        if(is_first_number){
            is_first_number = false;
            result = strtof(buffer, &buffer);
        }
        else {
            float number = strtof(buffer, &buffer);

            if(number == 0){
                strcpy(pointer_data_exchange, "-1");
                exit(-1);
            }
            result /= number;

            if (character == '\n' || character == EOF){
                is_first_number = true;
                while(strcmp(is_child,"0") == 0);
                sprintf(pointer_data_exchange, "%f", result);
                strcpy(is_child, "0");

            }
```

```
58          }
59      }
60
61      while(strcmp(is_child,"0") == 0);
62      strcpy(pointer_data_exchange, "\0");
63      strcpy(is_child, "0");
64
65      return 0;
66  }
```

Тест кейсы

Данные файла

```
1 2 5
4 4 4
5.31 5.11 1
5 5 1
23123 32 313 12
23123 32 313 3
```

Результат

```
machine@Turing:~/Desktop/MAI/os/lab4$ ./run
command
0.100000
0.250000
1.039139
1.000000
0.192384
0.769535
```

Вывод

В ходе лабороторной работы я познакомился с новым способом обмена данных между процессами посредством технологии «File mapping». Код со второй лабороторной работы послужил фундаментом, осталось заменить **pipe** на **shm_open** и **mmap** и организовать поочередную запись и чтение отображаемого участка памяти, это я осуществил так же с помощью технологии «File mapping». Подводя итог, могу сказать, что лично мне было приятней работать с **pipe**, так как в отличие от **mmap** не надо следить, закончил ли дочерний процесс свою работу, либо он продолжает делать вычисления. Однако, у **mmamp** есть свои плюсы, например прирост производительности по сравнению с обычной буферизированной работой с файлами