Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-207Б-23

Студент: Ивченко Матвей Сергеевич

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 2.11.2024

Москва, 2024.

Содержание

Постановка задачи

В рамках данной лабораторной работы необходимо составить и отладить программу на языке Си, которая выполняет работу с процессами и взаимодействие между ними в операционной системе. Основной процесс должен создать один дочерний процесс. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Также необходимо обрабатывать системные ошибки, которые могут возникнуть в процессе работы.

Общие сведения о программе

Программа состоит из двух файлов:

1. parent.c - основной файл, содержащий код родительского процесса.
2. child.c - файл, содержащий код дочернего процесса.

В родительском процессе используются следующие заголовочные файлы:

- `<stdio.h>` - для работы с стандартным вводом/выводом.
- `<stdlib.h>` - для работы с памятью и системными вызовами.
- `<unistd.h>` - для работы с системными вызовами.
- `<sys/mman.h>` - для работы с отображаемыми файлами.
- `<sys/stat.h>` - для работы с атрибутами файлов.
- `<sys/wait.h>` - для работы с процессами.
- `<fcntl.h>` - для работы с файловыми дескрипторами.
- `<string.h>` - для работы с строками.
- `<signal.h>` - для работы с сигналами.

В дочернем процессе используются следующие заголовочные файлы:

- `<stdio.h>` - для работы с стандартным вводом/выводом.
- `<stdlib.h>` - для работы с памятью и системными вызовами.
- `<string.h>` - для работы с строками.
- `<sys/mman.h>` - для работы с отображаемыми файлами.
- `<sys/stat.h>` - для работы с атрибутами файлов.
- `<fcntl.h>` - для работы с файловыми дескрипторами.
- `<unistd.h>` - для работы с системными вызовами.

Общий метод и алгоритм решения

Родительский процесс создает файл и устанавливает его размер.

1. Файл отображается в память с помощью `mmap`.

2. Родительский процесс создает дочерний процесс с помощью системного вызова `fork()`.
3. Дочерний процесс устанавливает обработчик сигнала `SIGUSR1` и ожидает его с помощью `pause()`.
4. Родительский процесс записывает введенные пользователем числа в отображаемый файл.
5. Родительский процесс отправляет сигнал `SIGUSR1` дочернему процессу.
6. Дочерний процесс выполняет вычисления (сложение чисел) и записывает результат в тот же файл.
7. Родительский процесс ожидает завершения дочернего процесса и освобождает отображаемую память.

Код программы

Содержимое файла parent.c:

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

#define FILEPATH "/tmp/file"
#define FILESIZE 4096

// ID дочернего процесса, который родитель будет использовать для отправки сигнала
pid_t child_pid;

// Обработчик сигнала, необходимый для дочернего процесса
void signal_handler(int sig) {
    // Ничего не делаем; просто ожидаем сигнала
}

int main() {
    int fd;
    char* mapped;

    // Открываем файл с правами на чтение и запись
```

```
    fd = open(FILEPATH, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("Error opening file.");
        exit(1);
    }

    // Устанавливаем размер файла
    if (ftruncate(fd, FILESIZE) == -1) {
        perror("Error setting file size.");
        close(fd);
        exit(1);
    }

    // Отображаем файл
    mapped = (char*)mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (mapped == MAP_FAILED) {
        perror("Error mapping file.");
        close(fd);
        exit(1);
    }
    close(fd);

    // Создаём дочерний процесс
    child_pid = fork();
    if (child_pid == -1) {
        perror("Fork failed.");
        munmap(mapped, FILESIZE);
        exit(1);
    } else if (child_pid == 0) {
        // Дочерний процесс: установка обработчика для SIGUSR1
        signal(SIGUSR1, signal_handler);

        // Ожидаем сигнал от родительского процесса
        pause();

        // Запускаем дочерний процесс через execve
        char filesize_str[10];
        sprintf(filesize_str, "%d", FILESIZE);
        const char* args[] = {"./child", FILEPATH, filesize_str, NULL};
```

```
        execve("./child", (char* const*)args, NULL);
        perror("execve failed.");
        exit(1);
    } else {
        // Родительский процесс

        // Ввод чисел
        char input_data[FILESIZE];
        printf("Введите числа через пробел: ");
        fgets(input_data, sizeof(input_data), stdin);

        // Копируем данные в отображаемый файл
        memcpy(mapped, input_data, FILESIZE);

        // Отправляем сигнал SIGUSR1 дочернему процессу
        kill(child_pid, SIGUSR1);

        // Ожидаем завершения дочернего процесса
        wait(NULL);

        // Освобождаем память
        munmap(mapped, FILESIZE);
    }

    return 0;
}
```

Содержимое файла child.c:

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

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Использование: %s <file_path> <int FILESIZE>\n", argv[0]);
        return 1;
    }
```

```
    // Получаем аргументы
    char* filepath = argv[1];
    int FILESIZE = atoi(argv[2]);

    // Открываем файл и отображаем его в память
    int fd = open(filepath, O_RDWR);
    if (fd == -1) {
        perror("Error opening file in child.");
        return 1;
    }

    char *mapped = (char*)mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (mapped == MAP_FAILED) {
        perror("Error mapping file in child.");
        close(fd);
        return 1;
    }
    close(fd);

    // Складываем числа
    int sum = 0;
    char *token = strtok(mapped, " ");
    while (token != NULL) {
        sum += atoi(token);
        token = strtok(NULL, " ");
    }

    // Записываем результат в тот же файл
    printf("Сумма чисел: %d\n", sum);

    // Освобождаем память
    munmap(mapped, FILESIZE);
    return 0;
}
```

Использование утилиты strace

На рисунке 1 приведён вывод strace, но только его часть, так как первая половина вызовов приходится на bash и запуск процессов.

```
openat(AT_FDCWD, "/tmp/file", O_RDWR|O_CREAT, 0600) = 3
ftruncate(3, 4096)                      = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7bad36aac000
close(3)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7bad36a96a10) = 20385
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\x61\x68\x61\x81\x92\x6c\x34\x33", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x61a9462bd000
brk(0x61a9462de000)                     = 0x61a9462de000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\32
0\265\321\200"..., 51Введите числа через пробел: ) = 51
read(0, 221 806 777
"221 806 777\n", 1024)          = 12
kill(20385, SIGUSR1)                    = 0
wait4(-1, Сумма чисел: 1804
NULL, 0, NULL)                = 20385
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=20385, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
munmap(0x7bad36aac000, 4096)            = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Разберём вывод:

1. openat открывает файл /tmp/file с определёнными правами доступа. Мы получаем файловый дескриптор 3;

2. ftruncate устанавливает размера файла дескриптора 3 равным 4096 байт;

3. mmap отображает файл в память

4. close закрывает файл дескриптор, так как он уже был отображён и теперь не нужен;

5. write выводит в стандартный поток вывода текст;

6. read читает данные из стандартного потока ввода;

7. kill отправляет сигнал дочернему процессу, что данные введены и можно их суммировать;

8. wait ждёт завершения дочернего процесса;

9. munmap освобождает отображённую память.

Демонстрация работы программы

```
matvey@matvey-M1502IA:~/workspace/MAI_OS_3sem2024$ ./parent
Введите числа через пробел: 221 806 777
Сумма чисел: 1804
matvey@matvey-M1502IA:~/workspace/MAI_OS_3sem2024$ ./parent
Введите числа через пробел: 1 2 3 4 5 6 7
Сумма чисел: 28
matvey@matvey-M1502IA:~/workspace/MAI_OS_3sem2024$ ./parent
Введите числа через пробел:
Сумма чисел: 0
```

Вывод

В данной лабораторной работе я освоил принципы работы с файловыми системами и обеспечении обмена данными между процессами с помощью «File mapping». Программа успешно выполняет свои задачи, создавая дочерний процесс, передавая ему данные через отображаемый файл и записывая результат в выходной файл. В целом, работа была интересной и полезной для понимания взаимодействия процессов в операционных системах.