Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа 3 по курсу

«Операционные системы»

III Семестр

Обеспечение обмена данных между процессами посредством технологии «File mapping»

Студент: Шляхтуров А.В
Группа: М8О-201Б-22
Преподаватель: Миронов Е. С.
Вариант 12
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Цель работы.

Приобретение практических навыков в освоении принципов работы с файловыми системами и обеспечении 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.

Решение

Родительский процесс создает первый дочерний процесс. Первый дочерний процесс создает второй дочерний процесс. Далее с помощью вызова shm_open мы создаем memory map файл. Родительский процесс отправляет считанные данные в дочерний процесс. Сначала мы получаем файловый дескриптор на начало участка памяти, куда мы спроецировали memory map файл. Этот файловый дескриптор возвращает вызов mmap. Таким образом один процесс может класть информацию в файл, а другой читать и переписывать. Дочерний процесс в свою очередь считывает информацию, переводит в верхний регистр и отправляет второму дочернему процессу через memory map файл. Второй дочерний процесс убирает все задвоенные пробелы и возвращает таким же образом данные в родителя. Родитель выводит в консоль. Для того, чтобы первый ребенок получил доступ после родителя, второй ребенок после первого, а родитель прочитал только после отправки данных вторым ребенком, мы используем три бинарных семафора.

Код

Main.c

```c
#include <sys/stat.h>
#include "stdio.h"
#include "stdlib.h"
#include <fcntl.h>
#include "unistd.h"
#include "sys/wait.h"
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <signal.h>
#include "../include/include.h"

int count_len_after_child2(char *buffer, int size)
{
    char result[1024];
    int i = 0;
    int j = 0;
    while (j < size)
    {
        if (buffer[j] == ' ' && buffer[j + 1] == ' ')
        {
            if (buffer[j + 2] != ' ')
            {
                result[i] = buffer[j + 2];
                i += 1;
                j += 3;
            }
            else
            {
                j += 2;
            }
        }
        else
        {
            result[i] = buffer[j];
            j += 1;
            i += 1;
        }
    }
    // printf("Result is: %s\n", result);
    return i;
}

int main()
{
    const int max_buffer_size = 1024;
    const int count_semafore = 3;
```

```
    // Инициализируем семафоры
    char *list_semafore_name[] = {"/semafor0", "/semafor1", "/semafor2"};

    sem_t *list_semafore[count_semafore];

    for (int i = 0; i < count_semafore; ++i)
    {
        if ((list_semafore[i] = sem_open(list_semafore_name[i], O_CREAT, S_IRWXU,
0)) == SEM_FAILED)
        {
            perror("semafore open in main");
        }
    }

    // Создаем файловый дескриптор

    int mmap_file_description;
    char mmap_file_name[] = "/mmap_file";

    // Создаем memory mapped файл, привязанный к дескриптору

    if ((mmap_file_description = shm_open(mmap_file_name, O_CREAT | O_RDWR, S_IR-
WXU)) == -1)
    {
        perror("open mmap_file_description");
        return -1;
    }

    pid_t child1 = create_processe();

    if (child1 == 0)
    {

        dup2FD(mmap_file_description, STDIN_FILENO);

        execl("../build/child1", list_semafore_name[0], list_semafore_name[1],
NULL);

        perror("child1");
        exit(EXIT_FAILURE);
    }

    pid_t child2 = create_processe();

    if (child2 == 0)
    {

        dup2FD(mmap_file_description, STDIN_FILENO);
        execl("../build/child2", list_semafore_name[1], list_semafore_name[2],
NULL);
```

```
        perror("child2");
        exit(EXIT_FAILURE);
    }

    char input_buffer[max_buffer_size];

    char enter[20] = "Enter a string: \n";

    write(1, enter, 20);

    char string[max_buffer_size];

    int count;

    char *mp;
    while ((count = read(0, string, max_buffer_size)) > 0)
    {
        // Установка размера memory-mapped файла
        if (ftruncate(mmap_file_descriptior, count) == -1)
        {
            perror("Could not set size");
            return 1;
        }

        // mp - указатель на начало отображенной области
        if ((mp = mmap(NULL, count, PROT_READ | PROT_WRITE, MAP_SHARED,
mmap_file_descriptior, 0)) == MAP_FAILED)
        {
            perror("mmap");
            return -1;
        }
        for (int i = 0; i < count; ++i)
        {
            mp[i] = string[i];
        }

        // printf("Даём команду первому ребенку\n");
        // fflush(stdout);

        if (sem_post(list_semafore[0]) == -1)
        {
            perror("Ошбика при отправке сигнала первому ребенку");
        }

        // printf("Ждём команду от второго ребенка\n");
        // fflush(stdout);

        if (sem_wait(list_semafore[2]) == -1)
        {
            perror("Ошбика при принятии сигнала от второго ребенка");
```

```
        }

        // printf("Получили команду от второго ребенка\n");
        // fflush(stdout);

        int len_after_child2 = count_len_after_child2(string, count);

        char result_buffer[max_buffer_size];

        for (int i = 0; i < len_after_child2; i++)
        {
            result_buffer[i] = mp[i];
        }

        printf("Печатаю результат:\n");
        printf("%s\n", result_buffer);
    }
    // удаляем запись о mmap file
    if ((unlink(mmap_file_name)) == -1)
    {
        perror("Ошибка при unlink");
    }

    if (kill(child1, SIGTERM) == -1)
    {
        perror("Ошибка при отправке сигнала");
    }

    if (kill(child2, SIGTERM) == -1)
    {
        perror("Ошибка при отправке сигнала");
    }

    for (int i = 0; i < count_semafore; ++i)
    {
        if ((sem_close(list_semafore[i])) == -1)
        {
            perror("Ошибка при закрытии семафора");
        }
        if ((sem_unlink(list_semafore_name[i])) == -1)
        {
            perror("Ошибка при удалении имени семафора");
        }
    }

    return 0;
}
```

Child1.c

```
#include <sys/stat.h>
#include "stdio.h"
```

```c
#include "stdlib.h"
#include <fcntl.h>
#include "unistd.h"
#include "sys/wait.h"
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <signal.h>
#include "../include/include.h"

#include "../include/include.h"

sem_t *child_list_semafor[2];

void handle_sigterm()
{
    for (int i = 0; i < 2; ++i)
    {
        closeFD(i);
        if ((sem_close(child_list_semafor[i])) == -1)
        {
            perror("Ошибка при закрытии семафора ch1");
        }
    }
    exit(0);
}

int main(int argc, char *argv[])
{
    // fflush(stdout);
    for (int i = 0; i < 2; ++i)
    {
        // printf("%s\n", argv[i]);
        if ((child_list_semafor[i] = sem_open(argv[i], O_CREAT)) == SEM_FAILED)
        {
            perror("semafore open in child1");
        }
    }

    struct stat sd;

    while (1)
    {
        // printf("Ждем команду от родителя\n");
        // fflush(stdout);

        if (sem_wait(child_list_semafor[0]) == -1)
        {
            perror("Ошбика при ожидании сигнала от родителя");
        }
```

```
        // printf("Получили команду от родителя\n");
        // fflush(stdout);

        if (signal(SIGTERM, handle_sigterm) == SIG_ERR)
        {
            perror("Error while setting a signal handler");
            return EXIT_FAILURE;
        }

        // читаем из shared memory, перенаправленного в STDIN
        if (fstat(STDIN_FILENO, &sd) == -1)
        {
            perror("could not get file size. \n");
        }
        char *mp;
        if ((mp = mmap(NULL, sd.st_size, PROT_READ | PROT_WRITE, MAP_SHARED,
STDIN_FILENO, 0)) == MAP_FAILED)
        {
            perror("mmap in child");
        }

        for (int i = 0; i < sd.st_size; ++i)
        {
            mp[i] = toupper(mp[i]);
        }

        // printf("Даём команду второму ребенку\n");
        if (sem_post(child_list_semafor[1]) == -1)
        {
            perror("Ошбика при отправке сигнала второму ребенку от первого");
        }
    }
}
```

Child2.c

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

#include "../include/include.h"

sem_t *child_list_semafor[2];
```

```
void handle_sigterm()
{
    for (int i = 0; i < 2; ++i)
    {
        closeFD(i);
        if ((sem_close(child_list_semafor[i])) == -1)
        {
            perror("Ошибка при закрытии семафора ch2");
        }
    }
    exit(0);
}

int main(int argc, char *argv[])
{
    // fflush(stdout);
    for (int i = 0; i < 2; ++i)
    {
        // printf("%s\n", argv[i]);
        if ((child_list_semafor[i] = sem_open(argv[i], O_CREAT)) == SEM_FAILED)
        {
            perror("semafore open in child1");
        }
    }

    struct stat sd;

    while (1)
    {
        // printf("Ждем команду от первого ребенка\n");
        // fflush(stdout);

        if ((sem_wait(child_list_semafor[0])) == -1)
        {
            perror("Ошбика при ожидании сигнала от ch1");
        }

        // printf("Получили команду от первого ребенка\n");
        // fflush(stdout);

        if (signal(SIGTERM, handle_sigterm) == SIG_ERR)
        {
            perror("Error while setting a signal handler");
            return EXIT_FAILURE;
        }

        // читаем из shared memory, перенаправленного в STDIN
        if (fstat(STDIN_FILENO, &sd) == -1)
        {
            perror("could not get file size. \n");
        }
```

```
        char *mp = mmap(NULL, sd.st_size, PROT_READ | PROT_WRITE, MAP_SHARED,
STDIN_FILENO, 0);

        char result[1024];
        int i = 0;
        int j = 0;
        while (j < sd.st_size)
        {
            if (mp[j] == ' ' && mp[j + 1] == ' ')
            {
                if (mp[j + 2] != ' ')
                {
                    result[i] = mp[j + 2];
                    i += 1;
                    j += 3;
                }
                else
                {
                    j += 2;
                }
            }
            else
            {
                result[i] = mp[j];
                j += 1;
                i += 1;
            }
        }

        for (int i = 0; i < sizeof(result); i++)
        {
            mp[i] = result[i];
        }

        // printf("Даём команду родителю\n");
        if ((sem_post(child_list_semafor[1])) == -1){
            perror("Ошбика при отправлении сигнала от ch1");
        }
    }
```

Пример работы

Input: aaa bb   cc dd

Output: AAABBCC DD

Выводы

Задание этой лабораторной работы очень похоже на задание первой, но здесь используется другой канал для обмена информацией между процессами. В первой лабе использовались каналы pipe, здесь же мы создаем memory map файл. Используя семафоры, мы можем создать очередность обращения каждого процесса к memory map файлу, чтобы не происходило одновременного чтения и записи информации в файл разными процессами.

Выполняя эту лабораторную работу, я научился пользоваться разделяемой памятью и понял принципы ее работы, использовал несколько новых системных вызовов и еще раз поработал с семафорами.