Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Агафонов А. C.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 13.11.24

Москва, 2024

**Постановка задачи**

**Вариант 20.**

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. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* **pid\_t fork(void)** — создаёт дочерний процесс. В программе используется для создания двух дочерних процессов, каждый из которых будет выполнять свою часть задачи.
* **int execl(const char \*path, const char \*arg, ...)** — загружает и исполняет новый образ программы. Дочерние процессы запускают программу child, передавая имя объекта разделяемой памяти в качестве аргумента.
* **int open(const char \*pathname, int flags, mode\_t mode)** — открытие/создание файла. В родительском процессе открываются два файла для записи результатов обработки данных дочерними процессами.
* **close(int fd)** — закрывает файл. Используется для закрытия всех открытых файлов в родительском и дочерних процессах после их использования.
* **int shm\_open(const char \*name, int oflag, mode\_t mode)** — создаёт или открывает разделяемую память. Родительский процесс создаёт два объекта разделяемой памяти для обмена данными с каждым из дочерних процессов.
* **int shm\_unlink(const char \*name)** — удаляет разделяемую память по имени. Используется для очистки системных ресурсов после завершения работы всех процессов.
* **int ftruncate(int fd, off\_t length)** — изменяет размер открытого файла. Применяется для установки размера разделяемой памяти, достаточного для размещения структуры данных.
* **void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)** — сопоставляет область памяти с файлом. Используется для отображения разделяемой памяти в адресное пространство процессов, чтобы они могли взаимодействовать с ней.
* **int munmap(void \*addr, size\_t length)** — отменяет сопоставление области памяти. Применяется для отмены отображения разделяемой памяти после завершения работы.
* **int sem\_post(sem\_t \*sem)** — сигнализирует (разблокирует) семафор. Родительский процесс использует sem\_post для подачи сигнала дочернему процессу, что данные готовы для обработки.
* **int sem\_wait(sem\_t \*sem)** — ожидает (блокируется) на семафоре. Дочерний процесс использует sem\_wait, чтобы дождаться сигнала от родителя перед началом обработки данных.

**Алгоритм работы программы**

Программа создает два дочерних процесса, используя fork(). Родительский процесс устанавливает общую память (shared memory) с помощью shm\_open и отображает её в адресное пространство с помощью mmap. Для синхронизации между процессами используются семафоры. Родительский процесс считывает строки ввода, распределяет их между дочерними процессами в зависимости от длины строки, а дочерние процессы обрабатывают данные (реверсируют строки) и записывают результат в соответствующие файлы.

**Код программы**

**main.cpp**

**#include <iostream>**

**#include <string>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <sys/mman.h>**

**#include <semaphore.h>**

**#include <sys/wait.h>**

**#include <cstring>**

**#include <cstdlib>**

**// Структура для обмена данными через отображаемый файл**

**struct shared\_data {**

**sem\_t sem\_parent; // Семафор для родительского процесса (сигнал о готовности данных)**

**sem\_t sem\_child;  // Семафор для дочернего процесса (сигнал о завершении обработки)**

**char buffer[1024];**

**int terminate;    // Флаг для завершения работы**

**};**

**int main() {**

**// Запрос имен файлов у пользователя**

**std::cout << "Enter file's name for child process 1: ";**

**std::string file1\_name;**

**std::getline(std::cin, file1\_name);**

**std::cout << "Enter file's name for child process 2: ";**

**std::string file2\_name;**

**std::getline(std::cin, file2\_name);**

**// Открытие файлов для записи**

**int file1 = open(file1\_name.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);**

**int file2 = open(file2\_name.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);**

**if (file1 < 0 || file2 < 0) {**

**perror("Can't open file");**

**exit(1);**

**}**

**// Создание отображаемых файлов (shared memory objects)**

**const char \*shm\_name1 = "/shm\_child1";**

**const char \*shm\_name2 = "/shm\_child2";**

**int shm\_fd1 = shm\_open(shm\_name1, O\_CREAT | O\_RDWR, 0666);**

**int shm\_fd2 = shm\_open(shm\_name2, O\_CREAT | O\_RDWR, 0666);**

**if (shm\_fd1 == -1 || shm\_fd2 == -1) {**

**perror("Can't create shared memory object");**

**exit(1);**

**}**

**// Установка размера отображаемых файлов**

**ftruncate(shm\_fd1, sizeof(shared\_data));**

**ftruncate(shm\_fd2, sizeof(shared\_data));**

**// Отображение файлов в память**

**shared\_data \*shm\_ptr1 = (shared\_data \*) mmap(NULL, sizeof(shared\_data), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd1, 0);**

**shared\_data \*shm\_ptr2 = (shared\_data \*) mmap(NULL, sizeof(shared\_data), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd2, 0);**

**if (shm\_ptr1 == MAP\_FAILED || shm\_ptr2 == MAP\_FAILED) {**

**perror("Can't mmap shared memory");**

**exit(1);**

**}**

**// Инициализация семафоров**

**sem\_init(&shm\_ptr1->sem\_parent, 1, 0);**

**sem\_init(&shm\_ptr1->sem\_child, 1, 0);**

**sem\_init(&shm\_ptr2->sem\_parent, 1, 0);**

**sem\_init(&shm\_ptr2->sem\_child, 1, 0);**

**// Установка флагов завершения работы**

**shm\_ptr1->terminate = 0;**

**shm\_ptr2->terminate = 0;**

**// Создание дочерних процессов**

**pid\_t pid1 = fork();**

**if (pid1 < 0) {**

**perror("Can't fork");**

**exit(1);**

**}**

**if (pid1 == 0) {**

**// Дочерний процесс 1**

**// Закрытие неиспользуемых отображаемых файлов**

**munmap(shm\_ptr2, sizeof(shared\_data));**

**close(shm\_fd2);**

**// Перенаправление стандартного вывода в файл**

**if (dup2(file1, STDOUT\_FILENO) < 0) {**

**perror("Can't redirect stdout for child process 1");**

**exit(1);**

**}**

**close(file1);**

**close(file2); // Не нужен в этом процессе**

**execl("./child", "./child", shm\_name1, NULL);**

**perror("Can't execute child process 1");**

**exit(1);**

**}**

**pid\_t pid2 = fork();**

**if (pid2 < 0) {**

**perror("Can't fork");**

**exit(1);**

**}**

**if (pid2 == 0) {**

**// Дочерний процесс 2**

**// Закрытие неиспользуемых отображаемых файлов**

**munmap(shm\_ptr1, sizeof(shared\_data));**

**close(shm\_fd1);**

**// Перенаправление стандартного вывода в файл**

**if (dup2(file2, STDOUT\_FILENO) < 0) {**

**perror("Can't redirect stdout for child process 2");**

**exit(1);**

**}**

**close(file2);**

**close(file1); // Не нужен в этом процессе**

**execl("./child", "./child", shm\_name2, NULL);**

**perror("Can't execute child process 2");**

**exit(1);**

**}**

**// Родительский процесс**

**close(file1);**

**close(file2);**

**while (true) {**

**std::string s;**

**std::getline(std::cin, s);**

**if (s.empty()) {**

**// Установка флагов завершения работы**

**shm\_ptr1->terminate = 1;**

**shm\_ptr2->terminate = 1;**

**// Сигнализация дочерним процессам**

**sem\_post(&shm\_ptr1->sem\_parent);**

**sem\_post(&shm\_ptr2->sem\_parent);**

**break;**

**}**

**if (s.size() > 10) {**

**// Отправка данных дочернему процессу 2**

**strcpy(shm\_ptr2->buffer, s.c\_str());**

**sem\_post(&shm\_ptr2->sem\_parent); // Сигнал дочернему процессу**

**sem\_wait(&shm\_ptr2->sem\_child);  // Ожидание завершения обработки**

**} else {**

**// Отправка данных дочернему процессу 1**

**strcpy(shm\_ptr1->buffer, s.c\_str());**

**sem\_post(&shm\_ptr1->sem\_parent); // Сигнал дочернему процессу**

**sem\_wait(&shm\_ptr1->sem\_child);  // Ожидание завершения обработки**

**}**

**}**

**// Ожидание завершения дочерних процессов**

**waitpid(pid1, NULL, 0);**

**waitpid(pid2, NULL, 0);**

**// Очистка ресурсов**

**sem\_destroy(&shm\_ptr1->sem\_parent);**

**sem\_destroy(&shm\_ptr1->sem\_child);**

**sem\_destroy(&shm\_ptr2->sem\_parent);**

**sem\_destroy(&shm\_ptr2->sem\_child);**

**munmap(shm\_ptr1, sizeof(shared\_data));**

**munmap(shm\_ptr2, sizeof(shared\_data));**

**close(shm\_fd1);**

**close(shm\_fd2);**

**shm\_unlink(shm\_name1);**

**shm\_unlink(shm\_name2);**

**return 0;**

**}**

**Child.cpp**

**#include <iostream>**

**#include <string>**

**#include <algorithm>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <sys/mman.h>**

**#include <semaphore.h>**

**#include <sys/stat.h>**

**#include <cstring>**

**#include <cstdlib>**

**// Структура для обмена данными через отображаемый файл**

**struct shared\_data {**

**sem\_t sem\_parent;**

**sem\_t sem\_child;**

**char buffer[1024];**

**int terminate;**

**};**

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

**if (argc != 2) {**

**std::cerr << "Usage: ./child <shm\_name>" << std::endl;**

**return 1;**

**}**

**const char \*shm\_name = argv[1];**

**// Открытие отображаемого файла**

**int shm\_fd = shm\_open(shm\_name, O\_RDWR, 0666);**

**if (shm\_fd == -1) {**

**perror("Can't open shared memory object");**

**exit(1);**

**}**

**// Отображение файла в память**

**shared\_data \*shm\_ptr = (shared\_data \*) mmap(NULL, sizeof(shared\_data), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);**

**if (shm\_ptr == MAP\_FAILED) {**

**perror("Can't mmap shared memory");**

**exit(1);**

**}**

**while (true) {**

**// Ожидание сигнала от родительского процесса**

**sem\_wait(&shm\_ptr->sem\_parent);**

**if (shm\_ptr->terminate) {**

**sem\_post(&shm\_ptr->sem\_child); // Сигнал родительскому процессу**

**break;**

**}**

**// Обработка данных**

**std::string str(shm\_ptr->buffer);**

**std::reverse(str.begin(), str.end());**

**// Запись результата в стандартный вывод**

**std::cout << str << std::endl;**

**std::cout.flush();**

**// Сигнал родительскому процессу о завершении обработки**

**sem\_post(&shm\_ptr->sem\_child);**

**}**

**munmap(shm\_ptr, sizeof(shared\_data));**

**close(shm\_fd);**

**return 0;**

**}**

**Протокол работы программы**

**Тестирование:**

root@1354a8b719e2:/workspaces/os\_base/lab3# ./main

Enter file's name for child process 1: one

Enter file's name for child process 2: two

123456789

12345678900

**Strace:**

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f5fae1a2000

mmap(NULL, 25258, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f5fae19b000

mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f5fadf2d000

mmap(0x7f5fadfd2000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa5000) = 0x7f5fadfd2000

mmap(0x7f5fae0fb000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7f5fae0fb000

mmap(0x7f5fae189000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25c000) = 0x7f5fae189000

mmap(0x7f5fae197000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f5fae197000

mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f5fade4e000

mmap(0x7f5fade5e000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7f5fade5e000

mmap(0x7f5faded1000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x83000) = 0x7f5faded1000

mmap(0x7f5fadf2b000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xdc000) = 0x7f5fadf2b000

mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f5fade21000

mmap(0x7f5fade25000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7f5fade25000

mmap(0x7f5fade48000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x27000) = 0x7f5fade48000

mmap(0x7f5fade4c000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2b000) = 0x7f5fade4c000

mmap(NULL, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f5fadc40000

mmap(0x7f5fadc66000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7f5fadc66000

mmap(0x7f5faddbb000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7f5faddbb000

mmap(0x7f5fade0e000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7f5fade0e000

mmap(0x7f5fade14000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f5fade14000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f5fadc3e000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f5fadc3c000

munmap(0x7f5fae19b000, 25258) = 0

openat(AT\_FDCWD, "/dev/shm/shm\_child1", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 5

openat(AT\_FDCWD, "/dev/shm/shm\_child2", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 6

mmap(NULL, 1096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f5fae1a1000

mmap(NULL, 1096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7f5fae1a0000

munmap(0x7f5fae1a1000, 1096) = 0

munmap(0x7f5fae1a0000, 1096) = 0

**Вывод**

При выполнении лабораторной работы возникли сложности с корректной инициализацией семафоров и синхронизацией между процессами, что приводило к дедлокам. Также было непросто отлаживать работу с общей памятью и убедиться, что все ресурсы правильно освобождаются после завершения работы. Было бы полезно иметь более подробную документацию по использованию mmap и семафоров в многопроцессных приложениях, а также примеры использования strace для отслеживания специфичных системных вызовов.