Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

# Лабораторная работа №3 по курсу

**«Операционные системы»**

**Процессы и потоки.**

**Взаимодействие между потоками.**

Студент: Модин-Глазков Богдан Арсеньевич

Группа: М8О–212Б–22

Вариант: 16

Преподаватель:

Оценка:

Дата:

Подпись:

Москва, 2023.

**Постановка задачи**

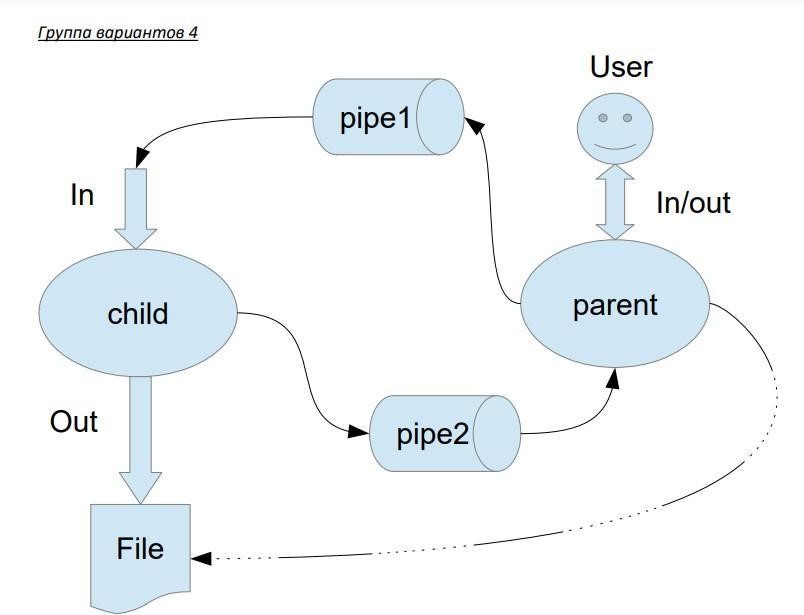
# Цель работы

Целью является 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.



Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись.

Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

13) Преобразовать верхний регистр в нижний, на месте пробелов поставить \_.

**Общие сведения о программе** Программа компилируется из файла main.c. Также используется заголовочные файлы: iostream, stdio.h, fcntl.h, unistd.h, sys/wait.h. sys/mman.h, sys/stat.h, string.h, string.

В программе используются следующие системные вызовы:

1. **mmap** – создает новое сопоставление в виртуальном адресном пространстве вызывающий процесс. Начальный адрес нового сопоставления: указан в addr. Аргументы функции: **void\* addr** — желаемый адрес начала участка отбраженной памяти, передаём 0 — тогда ядро само выберет этот адрес. **size\_t len** – количество байт, которое нужно отобразить в память, **int prot** — число, определяющее степень защищённости отображенного участка памяти(только чтение, только запись, исполнение, область недоступна). Обычные значения — PROT\_READ, PROT\_WRITE(можно кобминировать через ИЛИ), **int flag —** описывает атрибуты области. Обычное значение — MAP\_SHARED, **int filedes –** дескриптор файла, который нужно отобразить, **off\_t off –** смещение тображенного участка от начала файла
2. **munmap** – функция должна удалить любые сопоставления для всех страниц, содержащих любую часть адресного пространства процесса, начиная с addr и продолжая len байт. Аргументы функции: **void\* addr** – указатель на виртуальное адресное пространство, **size\_t len** - его размер в байтах.
3. **mremap** – функция переназначает адрес виртуальной памяти. Аргументы функции: **void\* old\_adress –** указатель на старое виртуальное адресное пространство, **size\_t old\_size** - старый размер блока виртуальной памяти, **size\_t new\_size –** требуемый размер блока виртуальной памяти, **unsigned long flags –** параметр, контролирующий работу с памятью (MREMAP\_MAYMOVE, MREMAP\_FIXED, MREMAP\_DONTUNMAP).

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap, malloc.
2. Переписать вариант первой лабараторной, работующей на pipe, используя mmap (munmap, mremap).
3. Реализовать простой интерфейс ввода и вывода результата.
4. Путем двух процессов работать со строками, сообщая между собой информацию.
5. В созданный по ходу работы с программой файл, записать строки, прошедшие на валидность.

**Основные файлы программы**

# main.cpp

|  |
| --- |
| #include <iostream> |
| #include <cstdlib> |
| #include <unistd.h> |
| #include <sys/types.h> |
| #include <sys/wait.h> |
| #include <sys/mman.h> |
| #include <fcntl.h> |
| #include <string> |
| #include <cstring> |
|  |
| using namespace std; |
|  |
| int main() { |
| string line; |
| getline(cin, line); |

|  |
| --- |
|  |
| const char \*filepath = "./data.bin"; |
| int fd = open(filepath, O\_RDWR | O\_CREAT | O\_TRUNC); // доступен для |
| записи и чтения, создастся если что и содержимое будет удалено |
|  |
| if (fd == -1) { |
| perror("Error opening file for writing"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| size\_t line\_size = line.size() \* sizeof(char); |
|  |
| ftruncate(fd, line\_size); |
|  |
| char \*map = (char \*) mmap(nullptr, line\_size + 1, PROT\_READ | PROT\_WRITE, |
| MAP\_SHARED, fd, 0); |
| if (map == MAP\_FAILED) { |
| close(fd); |
| perror("Error mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| for (size\_t i = 0; i < line\_size; i++) { |
| map[i] = line[i]; |
| } |
|  |

|  |
| --- |
| if (msync(map, line\_size, MS\_SYNC) == -1) { |
| perror("Could not sync the file to disk"); |
| } |
|  |
| pid\_t pid1 = fork(); |
| if (pid1 == -1) { |
| cerr << "Fork error" << endl; |
| return 1; |
| } |
|  |
| if (pid1 == 0) { |
| if (execl("./first\_child", "./first\_child") == -1) { |
| cerr << "Execl error" << strerror(errno) << endl; |
| return 1; |
| } |
| } |
|  |
| pid\_t pid2 = fork(); |
| if (pid2 == -1) { |
| cerr << "Fork error" << endl; |
| return 1; |
| } |
|  |
| if (pid2 == 0) { |
| if (execl("./second\_child", "./second\_child") == -1) { |
| cerr << "Execl error" << strerror(errno) << endl; |
| return 1; |
| } |
| } |
|  |
| wait(nullptr); |
|  |
| if (pid1 > 0 and pid2 > 0) { |
| write(STDOUT\_FILENO, map, line\_size); |
|  |
| if (munmap(map, line\_size + 1) == -1) { |
| close(fd); |
| perror("Error un-mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| close(fd); |
| } |
|  |
| return 0; |
| } |

first\_child.cpp

|  |
| --- |
| #include <iostream> |
| #include <cstdlib> |
| #include <unistd.h> |
| #include <sys/types.h> |

|  |
| --- |
| #include <sys/wait.h> |
| #include <sys/mman.h> |
| #include <fcntl.h> |
| #include <string> |
| #include <cstring> |
| #include <sys/stat.h> |
|  |
| using namespace std; |
|  |
| int main(int argc, const char \*argv[]) { |
|  |
| const char \*filepath = "./data.bin"; |
| int fd = open(filepath, O\_RDWR | O\_APPEND, 0666); |
|  |
| if (fd == -1) { |
| perror("Error opening file for writing"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| struct stat fileInfo = {0}; |
|  |
| if (fstat(fd, &fileInfo) == -1) { |
| perror("Error getting the file size"); |
| exit(EXIT\_FAILURE); |
| } |

|  |
| --- |
|  |
| ftruncate(fd, fileInfo.st\_size); |
|  |
| char \*map = (char \*) mmap(nullptr, fileInfo.st\_size + 1, PROT\_READ | |
| PROT\_WRITE, MAP\_SHARED, fd, 0); |
| if (map == MAP\_FAILED) { |
| close(fd); |
| perror("Error mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| char \*c = map; |
| for (size\_t i = 0; i < fileInfo.st\_size; ++i) { |
| \*c = tolower(\*c); |
| ++c; |
| } |
|  |
| if (msync(map, fileInfo.st\_size + 1, MS\_SYNC) == -1) { |
| perror("Could not sync the file to disk"); |
| } |
|  |
| if (munmap(map, fileInfo.st\_size + 1) == -1) { |
| close(fd); |
| perror("Error un-mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| close(fd); |
|  |
| return 0; |
| } |

second\_child.cpp

|  |
| --- |
|  |
| #include <iostream> |
| #include <cstdlib> |
| #include <unistd.h> |
| #include <sys/types.h> |
| #include <sys/wait.h> |
| #include <sys/mman.h> |
| #include <fcntl.h> |
| #include <string> |
| #include <cstring> |
| #include <sys/stat.h> |
|  |
| using namespace std; |
|  |
| int main(int argc, const char \*argv[]) { |
|  |
| const char \*filepath = "./data.bin"; |
| int fd = open(filepath, O\_RDWR | O\_APPEND, 0666); |

|  |
| --- |
|  |
| if (fd == -1) { |
| perror("Error opening file for writing"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| struct stat fileInfo = {0}; |
|  |
| if (fstat(fd, &fileInfo) == -1) { |
| perror("Error getting the file size"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| ftruncate(fd, fileInfo.st\_size); |
|  |
| char \*map = (char \*) mmap(nullptr, fileInfo.st\_size + 1, PROT\_READ | |
| PROT\_WRITE, MAP\_SHARED, fd, 0); |
| if (map == MAP\_FAILED) { |
| close(fd); |
| perror("Error mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| char \*c = map; |
| for (size\_t i = 0; i < fileInfo.st\_size; ++i) { |

|  |
| --- |
| if (\*c == ' ') { |
| \*c = '\_'; |
| } |
| ++c; |
| } |
|  |
| if (msync(map, fileInfo.st\_size + 1, MS\_SYNC) == -1) { |
| perror("Could not sync the file to disk"); |
| } |
|  |
| if (munmap(map, fileInfo.st\_size + 1) == -1) { |
| close(fd); |
| perror("Error un-mmapping the file"); |
| exit(EXIT\_FAILURE); |
| } |
|  |
| close(fd); |
|  |
| return 0; |
| } |

**Вывод**

Изучив принцип работы виртуальной памяти на низком уровне работы, я смог разобраться в ее работе и удобстве. Переписав первую лабораторную работу с технологии pipe на технологию mmap, я понял, что можно работать с передачей данных по-разному. Технология mmap очень удобна в работе, хоть и не так проста на первый взгляд. Важный аспект при работе с ней, грамотное выделение памяти с правами доступа и ее удаление, дабы избежать утечек и ошибок. В будущем мне может пригодиться умение работать с mmap, так как это очень актуальная технология на низкоуровневых разработках.