Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Эсмедляев Федор Романович

Группа: М8О–212Б–22

Вариант: 22

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

## 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 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.

Вариант 22) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Помимо этого, есть программа calculator.cpp, которая запускается при помощи exec. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **exec** - позволяет запустить исполняемый файл изнутри программы.
3. **open** - открытие файла
4. **сlose** - закрытие файла
5. **mmap -** создать маппинг файла
6. **munmap -** удалить маппинг файла
7. **sem\_open -** открыть семафор
8. **sem\_close -** закрыть семафор

**Общий метод и алгоритм решения**.

1. Программа main открывает файл для передачи данных и делает маппинг
2. Вводится семафор, чтобы заблокировать дочерний процесс
3. Происходит fork два раза, после чего идет разделение родительского и дочернего процесса
4. Родительский процесс читает данные из входного потока и записывает их в memory map.
5. После окончания записи семафор разблокируется и родительский процесс ожидает завершения дочернего.
6. Дочерний процесс запускает calculator при помощи exec
7. Дочерний процесс открывает и маппирует общий файл
8. Как только семафор разблокирован, процесс читает данные из мапа и производит вычисления
9. Вывод записывается в выходной файл

**Основные файлы программы**

**main.cpp:**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string>

int main()

{

int memoryd;

memoryd = open(

"memory.txt", // Имя файла

O\_RDWR | O\_CREAT | O\_TRUNC, // Флаги открытия файла (чтение и запись, создание файла, обрезать его до нулевой длины)

0666 // Режим доступа к файлу (права доступа)

);

ftruncate(memoryd, 1024);//обрезание memory.txt до 1024 байт

char\* buffer = (char\*)mmap(

NULL, // Адрес начала отображения (NULL означает, что ядро само выбирает адрес)

1024, // Размер отображаемой области в байтах

PROT\_READ | PROT\_WRITE, // Разрешения на чтение и запись

MAP\_SHARED, // Общий режим отображения (изменения видны другим процессам)

memoryd, // Файловый дескриптор, с которым связано отображение

0 // Смещение в файле (0 означает отображение с начала файла)

);

close(memoryd);

sem\_t\* sem = sem\_open("mmap\_sem", O\_CREAT, 0777, 0); // Open semaphore

//"mmap\_sem": Это строка, представляющая имя семафора. В данном случае, семафор называется "mmap\_sem".

//O\_CREAT: Флаг, указывающий, что семафор должен быть создан, если его нет. Если семафор существует, он будет открыт.

//0777: Это восьмеричное число, представляющее режим доступа к семафору. Здесь 0777 означает, что семафор доступен для чтения, записи и выполнения для всех пользователей.

//0: Это начальное значение семафора. В данном случае, семафор инициализируется значением 0.

if (sem == SEM\_FAILED)

{

perror("Could not open semaphore");

return -1;

}

// First fork

int id = fork();

if (id == -1)

{ // fork error

return 2;

}

else if (id == 0)

{ // First child process

// Second fork

int child\_id = fork();

if (child\_id == -1)

{ // fork error

return 2;

}

else if (child\_id == 0)

{ // Grandchild process

execl("./calculator", "./calculator", "mmap\_sem", NULL);

// "./calculator": Это имя исполняемого файла, который будет запущен с помощью execl.

// "./calculator": Это аргумент, передаваемый в запускаемую программу в качестве аргумента командной строки.

// "mmap\_sem": Это второй аргумент, передаваемый в запускаемую программу в качестве аргумента командной строки.

// NULL: Это последний аргумент для execl, который указывает на конец списка аргументов.

return 3;

}

else

{ // First child process

char c;

c = getchar();

int i = 0;

while (c != EOF)

{

buffer[i++] = c;

c = getchar();

}

buffer[i] = c;

sem\_post(sem); // Unlock semaphore

sem\_close(sem);

munmap(buffer, 1024);

int status;

waitpid(child\_id, &status, 0); // Waiting for the grandchild process to finish

if (status != 0)

perror("Grandchild process exited with an error");

return status;

}

}

else

{ // Parent process

// Second fork

int child\_id = fork();

if (child\_id == -1)

{ // fork error

return 2;

}

else if (child\_id == 0)

{ // Second child process

execl("./calculator", "./calculator", "mmap\_sem", NULL);

return 3;

}

else

{ // Parent process

waitpid(0, NULL, 0); // Waiting for the first child process to finish

waitpid(child\_id, NULL, 0); // Waiting for the second child process to finish

sem\_close(sem);

return 0;

}

}

}

**calculator.cpp:**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

#include <string>

#include <chrono>

#include <random>

std::mt19937 rng(std::chrono::steady\_clock::now().time\_since\_epoch().count());

std::string reverse(std::string& s)

{

std::string s1 = "";

for (int i = 0; i < s.size(); i++)

{

s1 += s[s.size() - i - 1];

}

return s1;

}

int main(int argc, char\*\* argv)

{

FILE\* f1 = fopen("output1.txt", "w");

FILE\* f2 = fopen("output2.txt", "w");

int memoryd;

memoryd = open("memory.txt", O\_RDWR, 0666);

char\* buffer = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, memoryd, 0);

close(memoryd);

sem\_t\* sem = sem\_open(argv[1], O\_CREAT, 0777, 0); // Open semaphore from argument list

if (sem == SEM\_FAILED)

{

perror("Could not open semaphore");

return -1;

}

size\_t i = 0;

sem\_wait(sem); // Wait for semaphore to unlock and then lock it

std::string s = "";

while (buffer[i] != -1)

{

if (buffer[i] != ' ' and buffer[i] != '\n')

{

s += buffer[i];

} else

if (buffer[i] == '\n')

{

s = reverse(s);

s = s + '\n';

if (rng() % 100 <= 80)

{

fputs(s.c\_str(), f1);

}

else

{

fputs(s.c\_str(), f2);

}

s = "";

}

i++;

}

sem\_close(sem);

munmap(buffer, 1024);

return 0;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(Memory\_Map)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(main main.cpp)

add\_executable(calculator calculator.cpp)

**Пример работы**

**1)**

**fedorubuntu@fedorubuntu-ZenBook-UX325JA-UX325JA:~/Документы/OS/lab3/build$ "/home/fedorubuntu/Документы/OS/lab3/build/main"**

**proverka**

**raz**

**dva**

**tri**

**chetiri**

**temp**

**lalal**

**abc**

**cba**

**output1.txt**

**akrevorp**

**zar**

**avd**

**irt**

**pmet**

**lalal**

**cba**

**abc**

**output2.txt**

**iritehc**

**2)**

fedorubuntu@fedorubuntu-ZenBook-UX325JA-UX325JA:~/Документы/OS/lab3/build$ "/home/fedorubuntu/Документы/OS/lab3/build/main"

"/home/fedorubuntu/Документы/OS/lab3/build/main"

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

proverka

raz

dva

tri

chetiri

temp

lalal

abc

cba

**output1.txt**

zar

avd

irt

pmet

cba

abc

akrevorp

avd

irt

iritehc

pmet

lalal

cba

abc

akrevorp

zar

avd

iritehc

pmet

lalal

abc

akrevorp

avd

irt

pmet

lalal

cba

abc

zar

avd

irt

pmet

cba

abc

akrevorp

zar

avd

irt

iritehc

pmet

lalal

cba

abc

**output2.txt**

akrevorp

iritehc

lalal

zar

irt

cba

zar

iritehc

akrevorp

iritehc

lalal

**Вывод**

В процессе выполнения работы я познакомился с таким инструментом, как memory map. Memory map создает виртуальную копию файла, что позволяет при работе с фалами не использовать дорогие операции чтения из файла и записи в файл. При использовании memory map ответственность за соответствие файла и его мапа переносится на операционную систему.