Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4 по курсу**

**«Операционные системы»**

Студент: Ворошилов Кирилл Сергеевич

Группа: М8О-201Б-21

Вариант: 18

Преподаватель Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/kiviyi/operation-system/tree/lab-4

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

**Задание**

Найти образец в строке наивным алгоритмом.

**Общие сведения о программе**

Как и во второй лабораторной.

**Общий метод и алгоритм решения**

Как и во второй лабораторной, только вместо pipe используется mmap.

**Исходный код**

#include "include/parent.h"

#include <vector>

int main() {

int n;

std::cin >> n;

std::vector <std::string> input;

std::string s;

getline(std::cin, s);

for (int i = 0; i < n; i++){

getline(std::cin, s);

input.push\_back(s);

}

// while (getline(std::cin, s)) {

// input.push\_back(s);

// }

std::vector <std::string> output = ParentRoutine(input);

for (const auto &res : output){

std::cout << res << std::endl;

}

return 0;

}

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

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

#include <sys/mman.h>

#include <unistd.h>

#include <iostream>

constexpr auto FIRST\_SHM\_NAME = "shared\_memory\_first"; // from parent to child1

constexpr auto SECOND\_SHM\_NAME = "shared\_memory\_second"; // from child2 to parent

constexpr auto THIRD\_SHM\_NAME = "third\_shared\_memory"; // from child1 to child2

constexpr auto FIRST\_SEMAP = "first\_semaphore";

constexpr auto SECOND\_SEMAP = "second\_semaphore";

constexpr auto THIRD\_SEMAP = "third\_semaphore";

std::vector<std::string> ParentRoutine(const std::vector<std::string> &input) {

char const \*pathToChild1 = "/home/kivi/os/lab\_4/l4/4child1";

char const \*pathToChild2 = "/home/kivi/os/lab\_4/l4/4child2";

std::vector<std::string> output;

int sfd1, semFd1;

createShm(sfd1, semFd1, FIRST\_SHM\_NAME, FIRST\_SEMAP);

makeFtruncateShm(sfd1, semFd1);

sem\_t \*sem1 = nullptr;

makeMmap((void \*\*) &sem1, PROT\_WRITE | PROT\_READ, MAP\_SHARED, semFd1);

sem\_init(sem1, 1, 0);

int sfd2, semFd2;

createShm(sfd2, semFd2, SECOND\_SHM\_NAME, SECOND\_SEMAP);

makeFtruncateShm(sfd2, semFd2);

sem\_t \*sem2 = nullptr;

makeMmap((void \*\*) &sem2, PROT\_WRITE | PROT\_READ, MAP\_SHARED, semFd2);

sem\_init(sem2, 1, 0);

int pid = fork();

if (pid == 0) { // child1

if (execl(pathToChild1, FIRST\_SHM\_NAME, FIRST\_SEMAP,

THIRD\_SHM\_NAME, THIRD\_SEMAP, nullptr) == -1) {

GetExecError(pathToChild1);

}

} else if (pid == -1) {

GetForkError();

} else {

char \*ptr;

makeMmap((void \*\*) &ptr, PROT\_READ | PROT\_WRITE, MAP\_SHARED, sfd1);

for (const std::string &s: input) {

sprintf((char \*) ptr, "%s", s.c\_str());

ptr += s.size() + 1;

sem\_post(sem1);

}

sprintf((char \*) ptr, "%s", "");

sem\_post(sem1);

int sfd3, semFd3;

createShm(sfd3, semFd3, THIRD\_SHM\_NAME, THIRD\_SEMAP);

makeFtruncateShm(sfd3, semFd3);

sem\_t \*sem3 = nullptr;

makeMmap((void \*\*) &sem3, PROT\_WRITE | PROT\_READ, MAP\_SHARED, semFd3);

sem\_init(sem3, 1, 0);

pid = fork();

if (pid == 0) { // child2

if (execl(pathToChild2, THIRD\_SHM\_NAME, THIRD\_SEMAP,

SECOND\_SHM\_NAME, SECOND\_SEMAP, nullptr) == -1) {

GetExecError(pathToChild2);

}

} else if (pid == -1) {

GetForkError();

} else { // parent

char \*ptr2;

makeMmap((void \*\*) &ptr2, PROT\_READ | PROT\_WRITE, MAP\_SHARED, sfd2);

while (true) {

sem\_wait(sem2);

std::string s = std::string(ptr2);

ptr2 += s.size() + 1;

if (s.empty()) {

break;

}

output.push\_back(s);

}

makeSemDestroy(sem1);

makeMunmap(sem1);

makeSemDestroy(sem2);

makeMunmap(sem2);

makeShmUnlink(FIRST\_SHM\_NAME);

makeShmUnlink(SECOND\_SHM\_NAME);

makeShmUnlink(FIRST\_SEMAP);

makeShmUnlink(SECOND\_SEMAP);

}

}

return output;

}

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

#include <sys/mman.h>

#include <unistd.h>

#include <fcntl.h>

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

if (argc != 4) {

std::cout << "Invalid arguments 1.\n";

exit(EXIT\_FAILURE);

}

int readFd, semInFd;

makeSharedMemoryOpen(readFd, argv[0], O\_CREAT | O\_RDWR, S\_IRWXU);

makeSharedMemoryOpen(semInFd, argv[1], O\_CREAT | O\_RDWR, S\_IRWXU);

int writeFd = 0, semOutFd = 0;

makeSharedMemoryOpen(writeFd, argv[2], O\_CREAT | O\_RDWR, S\_IRWXU);

makeSharedMemoryOpen(semOutFd, argv[3], O\_CREAT | O\_RDWR, S\_IRWXU);

char \*input, \*output;

sem\_t \*semInput, \*semOutput;

makeMmap((void \*\*) &input, PROT\_READ | PROT\_WRITE, MAP\_SHARED, readFd);

makeMmap((void \*\*) &output, PROT\_READ | PROT\_WRITE, MAP\_SHARED, writeFd);

makeMmap((void \*\*) &semInput, PROT\_READ | PROT\_WRITE, MAP\_SHARED, semInFd);

makeMmap((void \*\*) &semOutput, PROT\_READ | PROT\_WRITE, MAP\_SHARED, semOutFd);

char \*ptrIn = input, \*ptrOut = output;

while (true) {

sem\_wait(semInput);

std::string s = std::string(ptrIn);

ptrIn += s.size() + 1;

if (s.empty()) {

break;

}

for (char &ch: s) {

ch = tolower(ch);

}

sprintf((char \*) ptrOut, "%s", s.c\_str());

ptrOut += s.size() + 1;

sem\_post(semOutput);

}

sprintf((char \*) ptrOut, "%s", "");

sem\_post(semOutput);

makeMunmap(input);

makeMunmap(output);

makeMunmap(semInput);

makeMunmap(semOutput);

return 0;

}

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

#include <sys/mman.h>

#include <unistd.h>

#include <fcntl.h>

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

if (argc != 4) {

std::cout << "Invalid arguments 2.\n";

exit(EXIT\_FAILURE);

}

int readFd, semInFd;

makeSharedMemoryOpen(readFd, argv[0], O\_CREAT | O\_RDWR, S\_IRWXU);

makeSharedMemoryOpen(semInFd, argv[1], O\_CREAT | O\_RDWR, S\_IRWXU);

int writeFd, semOutFd;

makeSharedMemoryOpen(writeFd, argv[2], O\_CREAT | O\_RDWR, S\_IRWXU);

makeSharedMemoryOpen(semOutFd, argv[3], O\_CREAT | O\_RDWR, S\_IRWXU);

char \*input, \*output;

sem\_t \*semInput, \*semOutput;

makeMmap((void \*\*) &input, PROT\_READ | PROT\_WRITE, MAP\_SHARED, readFd);

makeMmap((void \*\*) &output, PROT\_READ | PROT\_WRITE, MAP\_SHARED, writeFd);

makeMmap((void \*\*) &semInput, PROT\_READ | PROT\_WRITE, MAP\_SHARED, semInFd);

makeMmap((void \*\*) &semOutput, PROT\_READ | PROT\_WRITE, MAP\_SHARED, semOutFd);

char \*ptrIn = input, \*ptrOut = output;

while (true) {

sem\_wait(semInput);

std::string s = std::string(ptrIn);

ptrIn += s.size() + 1;

if (s.empty() || s == " ") {

break;

}

int j = 0;

char lastCh = '\0';

for (size\_t i = 0; i < s.size(); i++){

if (lastCh != ' ' || s[i] != ' '){

s[j] = s[i];

j++;

}

lastCh = s[i];

}

std::string res;

for (int i = 0; i < j; i++) {

res += s[i];

}

sprintf((char \*) ptrOut, "%s", res.c\_str());

ptrOut += res.size() + 1;

sem\_post(semOutput);

}

sprintf((char \*) ptrOut, "%s", "");

sem\_post(semOutput);

makeMunmap(input);

makeMunmap(output);

makeMunmap(semInput);

makeMunmap(semOutput);

return 0;

}

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

#include <sys/mman.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

void makeSharedMemoryOpen(int &sfd, std::string name, int oflag, mode\_t mode) {

if ((sfd = shm\_open(name.c\_str(), oflag, mode)) == -1) {

std::cout << "Shm\_open error" << std::endl;

exit(EXIT\_FAILURE);

}

}

void makeMmap(void \*\*var, int prot, int flags, int fd) {

\*var = mmap(nullptr, getpagesize(), prot, flags, fd, 0);

if (var == MAP\_FAILED) {

std::cout << "Mmap error" << std::endl;

exit(EXIT\_FAILURE);

}

}

void makeSemDestroy(sem\_t \*sem) {

if (sem\_destroy(sem) == -1) {

std::cout << "Sem\_destroy error" << std::endl;

exit(EXIT\_FAILURE);

}

}

void makeMunmap(void \*addr) {

if (munmap(addr, getpagesize()) == -1) {

std::cout << "Munmap error" << std::endl;

exit(EXIT\_FAILURE);

}

}

void makeShmUnlink(std::string name) {

if (shm\_unlink(name.c\_str()) == -1) {

std::cout << "Shm\_unlink error" << std::endl;

exit(EXIT\_FAILURE);

}

}

void createShm(int &sfd, int &semInFd, const std::string &shmName,

const std::string &semap) {

makeSharedMemoryOpen(sfd, shmName, O\_CREAT | O\_RDWR, S\_IRWXU);

makeSharedMemoryOpen(semInFd, semap, O\_CREAT | O\_RDWR, S\_IRWXU);

}

void makeFtruncateShm(int &sfd, int &semInFd){

ftruncate(sfd, getpagesize());

ftruncate(semInFd, getpagesize());

}

void GetForkError() {

std::cout << "fork error" << std::endl;

exit(EXIT\_FAILURE);

}

void GetExecError(std::string const &executableFile) {

std::cout << "Exec \"" << executableFile << "\" error." << std::endl;

}

**Демонстрация работы программы**

kivi@LAPTOP-DGOM1IRE:~/os/lab\_4/l4$ ./lab4

DSAFDF ASDF

dsafdf asdf

**Выводы**

Чтобы проверить, реально ли ускоряют потоки программы я специально сделал два одинаковых input, отличающихся только количеством потоков. Из фотографий результата видно, что программа, работающая на 5 потоках значительно быстрее работает, чем программа работающая на 1 потоке... Также хочется отметить, что с потоками намного проще и интреснее работать, тк для их взаимодействий не нужно создавать pipe. Да и в целом потоки очень полезная, ускоряющая программу, штука.