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

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.