Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Операционные системы»

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

Студент: Чирикова П.С.
Преподаватель: Е.С. Миронов
Группа: М8О-201Б-21
Вариант: Дата:
Оценка:
Подпись:

Москва, 2023

1 Постановка задачи

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 решения 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. Перенаправление стандартных потоков Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

2 Сведения о программе

Программа написанна на Си в Unix подобной операционной системе на базе ядра Linux. При компиляции следует линковать библиотеки -lpthread и -lrt. В программе создаются дочерние процессы, данные в которые передаются с помощью shared memory.

При запуске программы пользователь вводит строки в стандартный поток ввода. Программа создает два дочерних процесса для преобразования введенных строк.

По завершении работы программа выводит в стандартный поток вывода введенные строки в верхнем регистре, удалив все задвоенные пробелы

3 Общий метод и алгоритм решения

Родительский процесс создает первый дочерний процесс, передав строки, полученные от пользователя. Затем родительский процесс создает второй дочерний процесс.

Первый дочерний прочесс принимает строки и приводит все символы в верхний регистр, после чего передавая полученные строки во второй дочерний процесс

Второй дочерний процесс принимает строки через pipe3, после чего удаляет все задвоенные пробелы и передает полученные строки родительскому процессу

Результирующие строки родительский процесс считывает от второго дочернего процесса

Все межпроцессорные взаимодействия реализованы через shared memory object. Для синхронизации работы процессов используются семафоры

4 Листинг программы

main.cpp

```
1#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("4child1", "4child2", input);

    for (const auto &res : output){
        std::cout << res << std::endl;
    }
    return 0;
}
```

parent.cpp

```
1#include "parent.h"
#include "utils.h"
#include <sys/mman.h>
#include <unistd.h>


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(char const *pathToChild1, char const *pathToChild2,
                                       const std::vector<std::string> &input) {

    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;
}
```

utils.cpp

```
#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;
}
```

child1.cpp

```
#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;

makeSharedMemory
Open(readFd,
argv[0], O_CREAT |
O_RDWR,
S_IRWXU);

makeSharedMemory
Open(semInFd,
argv[1], O_CREAT |
O_RDWR,
S_IRWXU);

    int writeFd = 0,
semOutFd = 0;

makeSharedMemory
Open(writeFd,
argv[2], O_CREAT |
O_RDWR,
S_IRWXU);

makeSharedMemory
Open(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 =
toupper(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(semIn
put);

makeMunmap(semO
utput);

    return 0;
```

child2.cpp

```
1#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;
}
```

5 Демонстрация работы программы

polina@polina-Vostro-3400:~/OS/tests$ cat lab4_test.cpp #include <gtest/gtest.h>

#include <array>
#include <memory>
#include <parent.h>
#include <vector>

```
TEST(FirstLabTests,SimpleTest) { constexpr int

inputSize = 3;

std::array<std::vector<std::string>,inputSize>input; input[0] = {
"abcabc",
"qwerty qwerty",
"A n O t H e R                    TeSt",
"oNe1 Two2 thr3ee 5fiVe                Ei8ght          13thiRTEEN                    ...",
"2 + 2 = 4",
"0123456789 abcdefghijklmnopqrstuvwxyz"
}; input[1] = {
"second test",
"1234567890/.,’][",
".                .                .                                    .....",
"!?+-*/_;",
}; input[2] = {
"AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
};

std::array<std::vector<std::string>,inputSize>expectedOutput; expectedOutput[0] = {
"ABCABC",
"QWERTY QWERTY",
"A N O T H E R TEST",
"ONE1 TWO2 THR3EE 5FIVE EI8GHT 13THIRTEEN ...",
"2 + 2 = 4",
"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ"
};
expectedOutput[1] = { "SECOND
TEST",
"1234567890/.,’][",
". . . .....",
"!?+-*/_;", };
expectedOutput[2] = {
"AAAAAAAAAAAAAAAA
AAAAAAAAAAA"
};
```

```
for (int i = 0; i <inputSize; i++) { auto result =
ParentRoutine(
"/home/botashev/ClionProjects/os_labs/cmake-build-debug/lab4/4child1",
"/home/botashev/ClionProjects/os_labs/cmake-build-debug/lab4/4child2", input[i]);
EXPECT_EQ(result,expectedOutput[i]);
} }
botashev@botashev-laptop:~/ClionProjects/os_labs/tests$ ./../cmake-build-debug/tests/l Running main()
from /home/botashev/ClionProjects/os_labs/cmake-build-debug/_deps/googl [==========] Running 1 test
from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from FirstLabTests
[ RUN      ] FirstLabTests.SimpleTest
[       OK ] FirstLabTests.SimpleTest (5 ms)
[----------] 1 test from FirstLabTests (5 ms total)

[----------] Global test environment tear-down [==========] 1 test
from 1 test suite ran. (5 ms total) [ PASSED ] 1 test.
```

6 Вывод

Взаимодействие между процессами можно организовать при помощи каналов, сокетов и отображаемых файлов. В данной лабораторной работе был изучен и применен механизм межпроцессорного взаимодействия – file mapping. Файл отображается на оперативную память таким образом, что мы можем взаимодействовать с ним как с массивом.

Благодаря этому вместо медленных запросов на чтение и запись мы выполняем отображение файла в ОЗУ и получаем произвольный доступ за O(1). Из-за этого при использовании этой технологии межпроцессорного взаимодействия мы можем получить ускорении работы программы, в сравнении, с использованием каналов.

Из недостатков данного метода можно выделить то, что дочерние процессы обязательно должны знать имя отображаемого файла и также самостоятельно выполнить отображение.