

**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: В. А. Амурский
Преподаватель: Е. С. Миронов
Группа: М8О-201Б-21
Вариант: 12
Дата:
Оценка:
Подпись:

Москва, 2023

1 Постановка задачи

Задача состоит в написании и отладке программы на языке Си, которая будет работать с процессами и обеспечивать взаимодействие между ними в одной из двух операционных систем. Целью программы является создание одного или нескольких дочерних процессов для решения задачи. Взаимодействие между процессами должно происходить через системные сигналы/события и/или каналы (pipe). Программа должна обрабатывать возможные системные ошибки.

Родительский процесс должен создавать два дочерних процесса, перенаправлять их стандартные потоки между собой дополнительным каналом и обеспечивать ввод пользовательских строк в pipe1. Child1 и Child2 должны работать над строками и передавать результаты своей работы родительскому процессу. Родительский процесс должен выводить полученные результаты в стандартный поток вывода. Программы родительского и дочерних процессов должны быть представлены отдельно.

2 Сведения о программе

Программа написана на C++ для Unix-подобной операционной системы, основанной на ядре Linux. 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 через pipe1 строки, полученные от пользователя. Затем родительский процесс создает второй дочерний процесс. Первый дочерний процесс преобразует строки, переводя их в верхний регистр, и передает их второму дочернему процессу через pipe3. Второй дочерний процесс удаляет все задвоенные пробелы и передает полученные строки родительскому процессу через pipe2. Родительский процесс считывает результирующие строки из pipe2 и выводит их в стандартный поток вывода.

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

4 Листинг программы

main.cpp

```

1 | #include "include/parent.h"
2 | #include <vector>
3 |
4 | int main() {
5 |     std::vector<std::string> input;
6 |     std::string s;
7 |
8 |     while (getline(std::cin, s)) {
9 |         input.push_back(s);
10 |    }
11 |
12 |    std::vector<std::string> output = ParentRoutine("4child1", "4child2", input);
13 |
14 |    for (const auto &res : output){
15 |        std::cout << res << std::endl;
16 |    }
17 |    return 0;
18 | }

```

parent.cpp

```

1 | #include "parent.h"
2 | #include "utils.h"
3 | #include <sys/mman.h>
4 | #include <unistd.h>
5 |
6 |
7 | constexpr auto FIRST_SHM_NAME = "shared_memory_first"; // from parent to child1
8 | constexpr auto SECOND_SHM_NAME = "shared_memory_second"; // from child2 to parent
9 | constexpr auto THIRD_SHM_NAME = "third_shared_memory"; // from child1 to child2
10 | constexpr auto FIRST_SEMAP = "first_semaphore";
11 | constexpr auto SECOND_SEMAP = "second_semaphore";
12 | constexpr auto THIRD_SEMAP = "third_semaphore";
13 |
14 |
15 | std::vector<std::string> ParentRoutine(char const *pathToChild1, char const *
    pathToChild2,
16 |                                     const std::vector<std::string> &input) {
17 |
18 |     std::vector<std::string> output;
19 |
20 |     int sfd1, semFd1;
21 |     createShm(sfd1, semFd1, FIRST_SHM_NAME, FIRST_SEMAP);
22 |     makeFtruncateShm(sfd1, semFd1);
23 |     sem_t *sem1 = nullptr;
24 |     makeMmap((void **) &sem1, PROT_WRITE | PROT_READ, MAP_SHARED, semFd1);
25 |     sem_init(sem1, 1, 0);
26 |
27 |
28 |     int sfd2, semFd2;

```

```

29 createShm(sfd2, semFd2, SECOND_SHM_NAME, SECOND_SEMAP);
30 makeFtruncateShm(sfd2, semFd2);
31 sem_t *sem2 = nullptr;
32 makeMmap((void **) &sem2, PROT_WRITE | PROT_READ, MAP_SHARED, semFd2);
33 sem_init(sem2, 1, 0);
34
35
36 int pid = fork();
37
38
39 if (pid == 0) { // child1
40     if (execl(pathToChild1, FIRST_SHM_NAME, FIRST_SEMAP,
41             THIRD_SHM_NAME, THIRD_SEMAP, nullptr) == -1) {
42         GetExecError(pathToChild1);
43     }
44 } else if (pid == -1) {
45     GetForkError();
46 } else {
47     char *ptr;
48     makeMmap((void **) &ptr, PROT_READ | PROT_WRITE, MAP_SHARED, sfd1);
49     for (const std::string &s: input) {
50         sprintf((char *) ptr, "%s", s.c_str());
51         ptr += s.size() + 1;
52         sem_post(sem1);
53     }
54     sprintf((char *) ptr, "%s", "");
55     sem_post(sem1);
56
57     int sfd3, semFd3;
58     createShm(sfd3, semFd3, THIRD_SHM_NAME, THIRD_SEMAP);
59     makeFtruncateShm(sfd3, semFd3);
60     sem_t *sem3 = nullptr;
61     makeMmap((void **) &sem3, PROT_WRITE | PROT_READ, MAP_SHARED, semFd3);
62     sem_init(sem3, 1, 0);
63
64
65     pid = fork();
66
67     if (pid == 0) { // child2
68         if (execl(pathToChild2, THIRD_SHM_NAME, THIRD_SEMAP,
69             SECOND_SHM_NAME, SECOND_SEMAP, nullptr) == -1) {
70             GetExecError(pathToChild2);
71         }
72     } else if (pid == -1) {
73         GetForkError();
74     } else { // parent
75         char *ptr2;
76         makeMmap((void **) &ptr2, PROT_READ | PROT_WRITE, MAP_SHARED, sfd2);
77

```

```

78         while (true) {
79             sem_wait(sem2);
80             std::string s = std::string(ptr2);
81             ptr2 += s.size() + 1;
82             if (s.empty()) {
83                 break;
84             }
85             output.push_back(s);
86         }
87
88
89         makeSemDestroy(sem1);
90         makeMunmap(sem1);
91
92
93         makeSemDestroy(sem2);
94         makeMunmap(sem2);
95
96         makeShmUnlink(FIRST_SHM_NAME);
97         makeShmUnlink(SECOND_SHM_NAME);
98         makeShmUnlink(FIRST_SEMAP);
99         makeShmUnlink(SECOND_SEMAP);
100     }
101 }
102 return output;
103 }

```

utils.cpp

```

1  #include "utils.h"
2  #include <sys/mman.h>
3  #include <semaphore.h>
4  #include <fcntl.h>
5  #include <unistd.h>
6
7
8  void makeSharedMemoryOpen(int &sfd, std::string name, int oflag, mode_t mode) {
9      if ((sfd = shm_open(name.c_str(), oflag, mode)) == -1) {
10         std::cout << "Shm_open error" << std::endl;
11         exit(EXIT_FAILURE);
12     }
13 }
14
15 void makeMmap(void **var, int prot, int flags, int fd) {
16     *var = mmap(nullptr, getpagesize(), prot, flags, fd, 0);
17     if (var == MAP_FAILED) {
18         std::cout << "Mmap error" << std::endl;
19         exit(EXIT_FAILURE);
20     }
21 }

```

```

22
23 void makeSemDestroy(sem_t *sem) {
24     if (sem_destroy(sem) == -1) {
25         std::cout << "Sem_destroy error" << std::endl;
26         exit(EXIT_FAILURE);
27     }
28 }
29
30 void makeMunmap(void *addr) {
31     if (munmap(addr, getpagesize()) == -1) {
32         std::cout << "Munmap error" << std::endl;
33         exit(EXIT_FAILURE);
34     }
35 }
36
37 void makeShmUnlink(std::string name) {
38     if (shm_unlink(name.c_str()) == -1) {
39         std::cout << "Shm_unlink error" << std::endl;
40         exit(EXIT_FAILURE);
41     }
42 }
43
44 void createShm(int &sfd, int &semInFd, const std::string &shmName,
45               const std::string &semmap) {
46     makeSharedMemoryOpen(sfd, shmName, O_CREAT | O_RDWR, S_IRWXU);
47     makeSharedMemoryOpen(semInFd, semmap, O_CREAT | O_RDWR, S_IRWXU);
48 }
49
50 void makeFtruncateShm(int &sfd, int &semInFd){
51     ftruncate(sfd, getpagesize());
52     ftruncate(semInFd, getpagesize());
53 }
54
55 void GetForkError() {
56     std::cout << "fork error" << std::endl;
57     exit(EXIT_FAILURE);
58 }
59
60 void GetExecError(std::string const &executableFile) {
61     std::cout << "Exec \"" << executableFile << "\" error." << std::endl;
62 }

```

child1.cpp

```

1 #include "utils.h"
2 #include <sys/mman.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5
6 int main(int argc, char *argv[]) {

```

```

7   if (argc != 4) {
8       std::cout << "Invalid arguments 1.\n";
9       exit(EXIT_FAILURE);
10  }
11
12  int readFd, semInFd;
13  makeSharedMemoryOpen(readFd, argv[0], O_CREAT | O_RDWR, S_IRWXU);
14  makeSharedMemoryOpen(semInFd, argv[1], O_CREAT | O_RDWR, S_IRWXU);
15
16  int writeFd = 0, semOutFd = 0;
17  makeSharedMemoryOpen(writeFd, argv[2], O_CREAT | O_RDWR, S_IRWXU);
18  makeSharedMemoryOpen(semOutFd, argv[3], O_CREAT | O_RDWR, S_IRWXU);
19
20
21  char *input, *output;
22  sem_t *semInput, *semOutput;
23  makeMmap((void **) &input, PROT_READ | PROT_WRITE, MAP_SHARED, readFd);
24  makeMmap((void **) &output, PROT_READ | PROT_WRITE, MAP_SHARED, writeFd);
25  makeMmap((void **) &semInput, PROT_READ | PROT_WRITE, MAP_SHARED, semInFd);
26  makeMmap((void **) &semOutput, PROT_READ | PROT_WRITE, MAP_SHARED, semOutFd);
27
28  char *ptrIn = input, *ptrOut = output;
29
30
31  while (true) {
32      sem_wait(semInput);
33      std::string s = std::string(ptrIn);
34      ptrIn += s.size() + 1;
35      if (s.empty()) {
36          break;
37      }
38      for (char &ch: s) {
39          ch = toupper(ch);
40      }
41
42      sprintf((char *) ptrOut, "%s", s.c_str());
43      ptrOut += s.size() + 1;
44      sem_post(semOutput);
45  }
46  sprintf((char *) ptrOut, "%s", "");
47  sem_post(semOutput);
48
49  makeMunmap(input);
50  makeMunmap(output);
51  makeMunmap(semInput);
52  makeMunmap(semOutput);
53
54  return 0;
55 }

```

child2.cpp

```
1 #include "utils.h"
2 #include <sys/mman.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5
6
7 int main(int argc, char *argv[]) {
8     if (argc != 4) {
9         std::cout << "Invalid arguments 2.\n";
10        exit(EXIT_FAILURE);
11    }
12
13    int readFd, semInFd;
14    makeSharedMemoryOpen(readFd, argv[0], O_CREAT | O_RDWR, S_IRWXU);
15    makeSharedMemoryOpen(semInFd, argv[1], O_CREAT | O_RDWR, S_IRWXU);
16
17    int writeFd, semOutFd;
18    makeSharedMemoryOpen(writeFd, argv[2], O_CREAT | O_RDWR, S_IRWXU);
19    makeSharedMemoryOpen(semOutFd, argv[3], O_CREAT | O_RDWR, S_IRWXU);
20
21
22    char *input, *output;
23    sem_t *semInput, *semOutput;
24    makeMmap((void **) &input, PROT_READ | PROT_WRITE, MAP_SHARED, readFd);
25    makeMmap((void **) &output, PROT_READ | PROT_WRITE, MAP_SHARED, writeFd);
26    makeMmap((void **) &semInput, PROT_READ | PROT_WRITE, MAP_SHARED, semInFd);
27    makeMmap((void **) &semOutput, PROT_READ | PROT_WRITE, MAP_SHARED, semOutFd);
28
29
30
31    char *ptrIn = input, *ptrOut = output;
32
33    while (true) {
34        sem_wait(semInput);
35        std::string s = std::string(ptrIn);
36        ptrIn += s.size() + 1;
37        if (s.empty() || s == " ") {
38            break;
39        }
40        int j = 0;
41        char lastCh = '\0';
42        for (size_t i = 0; i < s.size(); i++){
43            if (lastCh != ' ' || s[i] != ' '){
44                s[j] = s[i];
45                j++;
46            }
47            lastCh = s[i];
48        }
```



```

49
50     std::string res;
51     for (int i = 0; i < j; i++) {
52         res += s[i];
53     }
54
55     sprintf((char *) ptrOut, "%s", res.c_str());
56     ptrOut += res.size() + 1;
57     sem_post(semOutput);
58 }
59
60 sprintf((char *) ptrOut, "%s", "");
61 sem_post(semOutput);
62
63 makeMunmap(input);
64 makeMunmap(output);
65 makeMunmap(semInput);
66 makeMunmap(semOutput);
67 return 0;
68 }

```

5 Демонстрация работы программы

```

vazy1@vazy1-legion:~/ClionProjects/OS-labs/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 0 t H e R          TeSt",
    "oNe1 Two2 thr3ee    5fiVe      Ei8ght          13thiRTEEN          ...",
    "2 + 2 = 4",

```



```
Running main() from /home/botashev/ClionProjects/OS-labs/cmake-build-debug/_deps/goog
[=====] 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.