

Московский авиационный институт  
(Национальный исследовательский университет)  
Факультет "Информационные технологии и прикладная математика"  
Кафедра "Вычислительная математика и программирование"

**Лабораторная работа №4 по курсу  
“Операционные системы”**

*Студент:* Живалев Е.А.

*Группа:* М8О-206Б

*Преподаватель:* Соколов А.А.

*Вариант:* 27

*Оценка:* \_\_\_\_\_

*Дата:* \_\_\_\_\_

*Подпись:* \_\_\_\_\_

Москва, 2019

# 1 Задание

Программе на вход поступает число  $n$  - количество процессов, которое будет создано в ходе выполнения и программы, и  $n$  названий файлов, в которые эти процессы будут записывать данные. Родительский процесс создает  $n$  дочерних процессов, которым поочередно передает символы с входной строки. Дочерний процесс записывает полученные символ в переданный ему файл.

В ходе выполнения лабораторной работы были использованы следующие системные вызовы:

- `open` - открытие файла и получение его дескриптора
- `read` - использовался для чтения данных с входной строки
- `sem_open` - создание семафора или открытие ранее созданного
- `sem_post` - увеличение счетчика семафора
- `sem_wait` - уменьшение счетчика семафора
- `fork` - создание дочерних процессов
- `mmap` - отображение файла на память

## 2 Описание работы программы

С помощью функции `readLine`, которая, используя `read` считывает одну строку с входной строки, программа получает количество процессов (число  $n$ ), которые необходимо создать и имена файлов, в которые должна производиться запись. Затем создаётся  $n$  пайпов и  $n$  процессов. Затем в родительском процессе в цикле с условием, что возвращенное функцией `read` число больше 0 происходит запись считанного символа в определенный элемент массива `char`, который был записан в файл, отображенный в память, и увеличивает значение семафора, связанного с чтением. В то же время дочерний процесс считывает символ и записывает его в свой файл, а также увеличивает значение семафора, связанного с записью символов, тем самым давая родительскому процессу понять, что можно записать новый символ.

### 3 Исходный код

#### main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/stat.h>
7 #include <errno.h>
8 #include <stdlib.h>
9 #include <fcntl.h>
10 #include <sys/mman.h>
11 #include <semaphore.h>
12
13 #define FILE_NAME_LENGTH 129
14
15
16 void readLine(char* result) {
17     char c;
18     int byteCounter = 0;
19     while(byteCounter < FILE_NAME_LENGTH) {
20         if(read(STDIN_FILENO, &c, 1) == -1) {
21             perror("Reading failed");
22         }
23         if(c == '\n') {
24             break;
25         } else {
26             result[byteCounter++] = c;
27         }
28     }
29     result[byteCounter] = '\0';
30 }
31
32 int main() {
33     pid_t pid;
34     char c;
35     int numberOfProcesses = 1;
36     char buf[FILE_NAME_LENGTH];
37     readLine(buf);
38     char* inputSemaphoreName = strdup("/semaphore?input");
39     char* outputSemaphoreName = strdup("/semaphore?output");
40     if(atoi(buf) > 0) {
41         numberOfProcesses = atoi(buf);
42     } else {
43         printf("Invalid argument - must be a positive number\n");
44         exit(-1);
45     }
46     char* map;
47     char fileNames[numberOfProcesses][FILE_NAME_LENGTH];
48     int processNumber = 0;
49     sem_t* inputSemaphores[numberOfProcesses];
50     sem_t* outputSemaphores[numberOfProcesses];
51     for(int i = 0; i < numberOfProcesses; ++i) {
52         readLine(buf);
53         strcpy(fileNames[i], buf);
54     }
55     for(int i = 0; i < numberOfProcesses; ++i) {
56         inputSemaphoreName[10] = '0' + i;
```

```

57     outputSemaphoreName[10] = '0' + i;
58     inputSemaphores[i] = sem_open(inputSemaphoreName, O_CREAT,
777, 0);
59     outputSemaphores[i] = sem_open(outputSemaphoreName,
O_CREAT, 777, 1);
60     if(inputSemaphores[i] == SEM_FAILED ||
61         outputSemaphores[i] == SEM_FAILED) {
62         perror("Fault during semaphore init");
63         return -1;
64     }
65     sem_unlink(inputSemaphoreName);
66     sem_unlink(outputSemaphoreName);
67 }
68 char* tempFileName = strdup("/tmp/tmp_file.XXXXXX");
69 int tempFileDescriptor = mkstemp(tempFileName);
70 free(tempFileName);
71 int fileSize = numberOfProcesses + 1;
72 char temp[fileSize];
73 for(int i = 0; i < fileSize; ++i) {
74     temp[i] = ' ';
75 }
76 write(tempFileDescriptor, temp, fileSize);
77 if((map = (char*)mmap(NULL, fileSize, PROT_WRITE | PROT_READ,
MAP_SHARED,
78     tempFileDescriptor, 0)) == MAP_FAILED) {
79     perror("Mapping failed");
80     return -1;
81 }
82 for(int i = 0; i < numberOfProcesses; ++i) {
83     pid = fork();
84     if(pid == 0) {
85         processNumber = i;
86         break;
87     } else if (pid == -1) {
88         perror("Fork failed");
89         return -1;
90     }
91 }
92 if(pid > 0) {
93     while(read(STDIN_FILENO, &c, 1) > 0) {
94         sem_wait(outputSemaphores[processNumber]);
95         map[processNumber] = c;
96         sem_post(inputSemaphores[processNumber]);
97         processNumber++;
98         processNumber %= numberOfProcesses;
99     }
100     for(int i = 0; i < numberOfProcesses; ++i) {
101         map[i] = '\0';
102         sem_post(inputSemaphores[i]);
103     }
104 } else if(pid == 0) {
105     int fd = open(fileNames[processNumber], O_CREAT | O_WRONLY
, S_IREAD |
106     S_IWRITE);
107     if(ftruncate(fd, 0) == -1) {
108         perror("truncation failed");
109     }
110     while(1) {
111         sem_wait(inputSemaphores[processNumber]);

```

```

112         char c;
113         c = map[processNumber];
114         sem_post(outputSemaphores[processNumber]);
115         if(c == '\0') {
116             break;
117         } else {
118             if(write(fd, &c, 1) == -1) {
119                 perror("Writing failed");
120             }
121         }
122     }
123     close(fd);
124     exit(0);
125 }
126 close(tempFileDescriptor);
127 munmap(map, fileSize);
128 return 0;
129 }

```

## 4 Консоль

```
qelderdelta@qelderdelta-UX331UA:~/Study/OS/os_lab_4/src$ ./a.out
3
1.txt
2.txt
3.txt
ttthhhiiisss  ssshhoouuulllddd  bbbeee  iinmn  fffiiilllee123
qelderdelta@qelderdelta-UX331UA:~/Study/OS/os_lab_4/src$ cat 1.txt
this should be in file1
qelderdelta@qelderdelta-UX331UA:~/Study/OS/os_lab_4/src$ cat 2.txt
this should be in file2
qelderdelta@qelderdelta-UX331UA:~/Study/OS/os_lab_4/src$ cat 3.txt
this should be in file3
```

## 5 Выводы

В ходе выполнения лабораторной работы я познакомился с таким интересным механизмом, как отображение файла в память, который позволяет выиграть в производительности по сравнению с обычным чтением из файла за счет уменьшения количества системных вызовов, а также лишнего копирования.