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: Боев Савелий Сергеевич

Группа: М8О-207Б-21

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

[Репозиторий 3](#_Toc123904211)

[Постановка задачи 3](#_Toc123904212)

[Цель работы 3](#_Toc123904213)

[Задание 3](#_Toc123904214)

[Общие сведения о программе 4](#_Toc123904215)

[Общий метод и алгоритм решения 7](#_Toc123904216)

[Исходный код 8](#_Toc123904217)

[Демонстрация работы программы 8](#_Toc123904218)

[Выводы 13](#_Toc123904219)

# **Репозиторий**

https://github.com/IamNoobLEL/Labs-OSi

# **Постановка задачи**

## **Цель работы**

Изучение операционных систем

## **Задание**

Реализовать программу, в которой родительский процесс создает один дочерний процесс. Родительский процесс принимает путь к файлу и строки, которые отправляются в тот дочерний процесс, там те из них, которые оканчиваются на символы ‘;’ или ‘.’, записываются в файл, если же строки не удовлетворяют этому правилу, то они возвращаются в родительский процесс. Далее в родительском процессе сначала выводятся строки из файла (если его удалось открыть и там есть хотя бы одна строка), а потом строки, вернувшиеся из родительского процесса в дочерний.

# **Общие сведения о программе**

В программе используются следующие библиотеки:

* <stdio.h> - для вывода информации на консоль
* <fcntl.h> - для работы с файлами
* <unsitd.h> - для системных вызовов fork, ftruncate и close в Ubuntu
* <sys/wait.h> - для функции waitpid, когда родительский процесс ждёт дочерний
* <sys/mman.h> - для вызова функций, использующихся для операций с отображаемыми файлами (mmap, mremap, munmap)
* <sys/stat.h> - для вызова функции fstat
* <string.h> - для вызова функции strcpy
* <string> - для работы со строками в формате C++

В задании используются такие команды и строки, как:

* **void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)** **–** команда, отображающая файл на физическую память (ОЗУ) и принимающая 6 аргументов:
  + Адрес начала сопоставления (у меня в программе везде NULL, то есть ядро операционной системы само выбирает адрес).
  + Длина (в байтах).
  + Аргументы PROT (у меня везде PROT\_READ | PROT\_WRITE, то есть отображение можно прочесть, а можно и записать в него информацию).
  + Флаги, означающие, видны ли отображения другим процессам (у меня везде есть MAP\_SHARED – обновления видны всем процессам, имеющим данное представление отображаемого файла, также если присутствует флаг MAP\_ANONYMOUS, то следующий, 5 аргумент игнорируется).
  + Файловый дескриптор – если не указан флаг MAP\_ANONYMOUS в предыдущем, 4 аргументе, то при записи в отображаемый файл информация будет записываться в файл, дескриптор когорого указан в данном аргументе.
  + Смещение на указанное количество байт от начальной позиции, должно быть кратно размеру страницы (у меня смещений нет, так что везде указано 0).

В случае успеха функция возвращает указатель на отображаемый файл, в противном случае – указатель типа void на -1.

* **int open(const char \*pathname, int flags, mode\_t mode) –** функция, открывающая файл и принимающая 3 аргумента:
  + Путь к файлу.
  + Флаги (в моём случае это O\_RDWR | O\_CREAT, означающие, что в файл можно записывать, а также читать, и если файла не существовало, то создать его).
  + Параметры (в моём случае самые “разрешающие” параметры – S\_IRWXU | S\_IRWXG | S\_IRWXO, то есть файл разрешено читать и записывать в него всем пользователям).

В случае успеха вернёт число, означающее файловый дескриптор, в противном случае будет возвращено -1.

* **int munmap(void \*addr, size\_t length)** **–** функция, удаляющая отображение из заданной области и принимающая 2 аргумента:
  + Указатель на отображаемый файл.
  + Размер отображаемого файла.

В случае успеха возвращает значение 0, в противном случае возвращает -1.

* **int ftruncate(int fd, off\_t length) –** функция, урезающая файл до указанной длины и принимающая 2 аргумента:
  + Файловый дескриптор.
  + Новая длина файла.

В случае успеха вернётся значение 0, в противном случае -1.

* **void \*mremap(void \*old\_address, size\_t old\_size, size\_t new\_size, int flags, ... /\* void \*new\_address \*/) –** функция, изменяющая размер и возможно адрес отображаемого файла и принимающая 4 или 5 аргументов:
  + Указатель на старый адрес виртуальной памяти.
  + Старый размер отображаемого файла.
  + Новый размер отображаемого файла.
  + Флаги (у меня присутствует флаг MREMAP\_MAYMOVE, означающий, что если изменить размер памяти, находясь на текущем адресе нельзя, то ядро операционной системы может поменять адрес, однако в таком случае остальные указатели на данную область памяти становятся невалидными, но у меня никаких разных указателей на одинаковые куски памяти нет, поэтому я могу смело использовать этот флаг).
  + Новый адрес памяти, если в предыдущем, 4 пункте, указан флаг MREMAP\_FIXED, то тогда можно указать новый адрес памяти, куда будет перемещено отображение файла (у меня нигде 5 аргумент функции mremap не используется).

В случае успеха функция вернёт указатель на новый участок памяти.

# **Общий метод и алгоритм решения**

В начале программа получает на вход путь к файлу, где будут лежать нужные строки, затем пользователю предлагается ввести строки, конец ввода должен сигнализироваться символов Ctrl+D (это некоторое количество строк считывается как одна (с символами переноса строки и символом конца строки – ‘\0’). Далее программа создаёт дочерний процесс, если этого сделать не удалось, то она аварийно завершается.

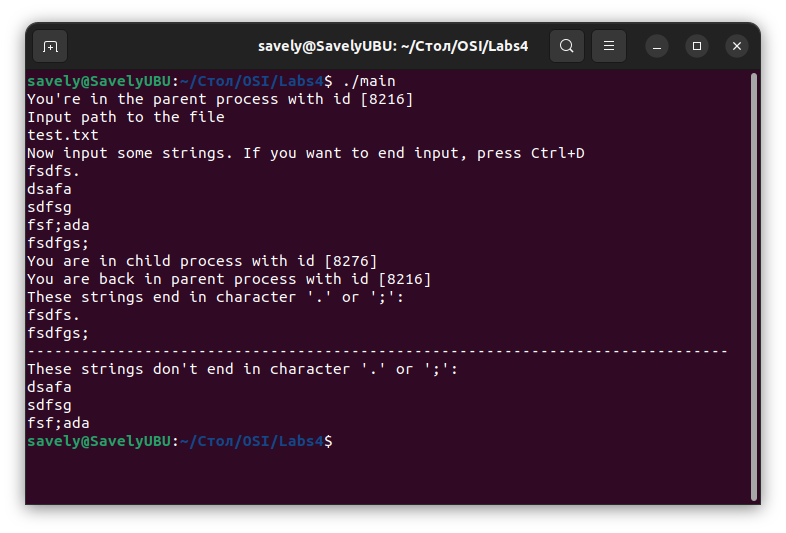
В дочернем процессе получается имя файла, а также большая исходная строка. В результате прохода по строке и некоторых операций получаются 2 строки file\_string и out\_string. Если удалость открыть файл, то file\_string записывается в файл, out\_string по неименованному каналу передаётся обратно в родительский процесс, вся память чистится, закрываются файловые дескрипторы и возвращается значение 0. Если же файл открыть не удалось, то выводится информация об ошибке открытия файла, и возвращается значение 1.

Родительский процесс сразу, как только запускается, ждёт дочерний. Если получаемое значение 1, то программа завершается, предварительно очистив всю память и закрыв файловые дескрипторы. Если же возвращаемое значение 0, то тогда программа пытается открыть файл. Если это не получилось сделать, то она аварийно завершается. В противном случае она читает из файла строки и выводит их, предварительно написав, что это строки, удовлетворяющие правилу. Дальше выводятся строки, не удовлетворяющие правилу. Далее чистится вся память и удаляются файловые дескрипторы.

# **Исходный код**

1. #include <stdio.h>
2. #include <fcntl.h>
3. #include <unistd.h>
4. #include <sys/wait.h>
5. #include <sys/mman.h>
6. #include <sys/stat.h>
7. #include <string.h>
8. #include <string>
9. int main()
10. {
11. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("You're in the parent process with id [%i]**\n**", getpid());
12. char symbol, \*in = (char \*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char)), \*file\_path = (char \*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char));
13. int \*size = (int \*)mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, 0, 0), counter = 0;
14. if (size == MAP\_FAILED)
15. {
16. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Mapping failed in creation of integer value**\n**");
17. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
18. }
19. \*size = 1;
20. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Input path to the file**\n**");
21. while ((symbol = [getchar](http://www.opengroup.org/onlinepubs/009695399/functions/getchar.html)()) != '**\n**')
22. {
23. file\_path[counter++] = symbol;
24. if (counter == \*size)
25. {
26. \*size \*= 2;
27. file\_path = (char \*)[realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html)(file\_path, (\*size) \* sizeof(char));
28. }
29. }
30. file\_path = (char \*)[realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html)(file\_path, (counter + 1) \* sizeof(char));
31. file\_path[counter] = '**\0**';
32. counter = 0, \*size = 1;
33. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Now input some strings. If you want to end input, press Ctrl+D**\n**");
34. while ((symbol = [getchar](http://www.opengroup.org/onlinepubs/009695399/functions/getchar.html)()) != EOF)
35. {
36. in[counter++] = symbol;
37. if (counter == \*size)
38. {
39. \*size \*= 2;
40. in = (char \*)[realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html)(in, (\*size) \* sizeof(char));
41. }
42. }
43. \*size = counter + 1;
44. in = (char \*)[realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html)(in, (\*size) \* sizeof(char));
45. in[(\*size) - 1] = '**\0**';
46. char \*ptr = (char \*)mmap(NULL, (\*size) \* sizeof(char), PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, 0, 0);
47. if (ptr == MAP\_FAILED)
48. {
49. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Mapping failed in creation of array of chars**\n**");
50. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
51. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
52. int err = munmap(size, sizeof(int));
53. if (err != 0)
54. {
55. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
56. }
57. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
58. }
59. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(ptr, in);
60. int fd = open(file\_path, O\_RDWR | O\_CREAT, S\_IRWXU | S\_IRWXG | S\_IRWXO);
61. if (fd == -1)
62. {
63. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to open file**\n**");
64. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
65. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
66. int err1 = munmap(ptr, (\*size) \* sizeof(char));
67. int err2 = munmap(size, sizeof(int));
68. if ((err1 != 0) || (err2 != 0))
69. {
70. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
71. }
72. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
73. }
74. char \*f = (char \*)mmap(NULL, sizeof(char), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);
75. if (f == MAP\_FAILED)
76. {
77. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to create string associated with file**\n**");
78. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
79. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
80. int err1 = munmap(ptr, (\*size) \* sizeof(char));
81. int err2 = munmap(size, sizeof(int));
82. if ((err1 != 0) || (err2 != 0))
83. {
84. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
85. }
86. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
87. }
88. pid\_t child\_pid = fork();
89. if (child\_pid == -1)
90. {
91. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to create child process**\n**");
92. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
93. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
94. int err1 = munmap(ptr, (\*size) \* sizeof(char));
95. int err2 = munmap(size, sizeof(int));
96. if ((err1 != 0) || (err2 != 0))
97. {
98. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
99. }
100. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
101. }
102. else if (child\_pid == 0)
103. {
104. *//child*
105. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("You are in child process with id [%i]**\n**", getpid());
106. std::string string = std::string(), file\_string = std::string(), out\_string = std::string();
107. for (int i = 0; i < \*size; i++)
108. {
109. if (i != (\*size) - 1)
110. {
111. string += ptr[i];
112. }
113. if ((ptr[i] == '**\n**') || (i == (\*size) - 1))
114. {
115. if ((i > 0) && ((ptr[i - 1] == '.') || (ptr[i - 1] == ';')))
116. {
117. file\_string += string;
118. }
119. else
120. {
121. out\_string += string;
122. }
123. string = std::string();
124. }
125. }
126. if ((file\_string.length()) && (file\_string[file\_string.length() - 1] != '**\n**'))
127. {
128. file\_string += '**\n**';
129. }
130. if (file\_string.length() != 0)
131. {
132. if ((ftruncate(fd, std::max((int)file\_string.length(), 1) \* sizeof(char))) == -1)
133. {
134. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to truncate file**\n**");
135. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
136. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
137. return 1;
138. }
139. if ((f = (char \*)mremap(f, sizeof(char), (file\_string.length() + 1) \* sizeof(char), MREMAP\_MAYMOVE)) == ((void \*)-1))
140. {
141. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to resize memory for string associated with file**\n**");
142. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
143. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
144. return 1;
145. }
146. [sprintf](http://www.opengroup.org/onlinepubs/009695399/functions/sprintf.html)(f, "%s", file\_string.c\_str());
147. }
148. if ((out\_string.length()) && (out\_string[out\_string.length() - 1] != '**\n**'))
149. {
150. out\_string += '**\n**';
151. }
152. if ((ptr = (char \*)mremap(ptr, (\*size) \* sizeof(char), out\_string.length() + 1, MREMAP\_MAYMOVE)) == ((void \*)-1))
153. {
154. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Failed to truncate file for string**\n**");
155. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
156. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
157. return 1;
158. }
159. \*size = out\_string.length() + 1;
160. [sprintf](http://www.opengroup.org/onlinepubs/009695399/functions/sprintf.html)(ptr, "%s", out\_string.c\_str());
161. }
162. else
163. {
164. *//parent*
165. int wstatus;
166. waitpid(child\_pid, &wstatus, 0);
167. if (wstatus)
168. {
169. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
170. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
171. int err1 = munmap(ptr, (\*size) \* sizeof(char));
172. int err2 = munmap(f, counter \* sizeof(char));
173. int err3 = munmap(size, sizeof(int));
174. if ((err1 != 0) || (err2 != 0) || (err3 != 0))
175. {
176. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
177. }
178. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
179. }
180. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("You are back in parent process with id [%i]**\n**", getpid());
181. struct stat statbuf;
182. if (fstat(fd, &statbuf) < 0)
183. {
184. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Problems with opening file %s**\n**", file\_path);
185. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
186. }
187. counter = std::max((int)statbuf.st\_size, 1);
188. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("These strings end in character '.' or ';':**\n**");
189. if (statbuf.st\_size > 1)
190. {
191. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s", f);
192. }
193. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("------------------------------------------------------------------------------**\n**These strings don't end in character '.' or ';':**\n**");
194. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s", ptr);
195. close(fd);
196. int err1 = munmap(ptr, (\*size) \* sizeof(char));
197. int err2 = munmap(f, counter \* sizeof(char));
198. int err3 = munmap(size, sizeof(int));
199. if ((err1 != 0) || (err2 != 0) || (err3 != 0))
200. {
201. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unmapping failed**\n**");
202. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
203. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
204. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
205. }
206. }
207. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
208. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(file\_path);
209. return 0;
210. }

## **Демонстрация работы программы**



# **Выводы**

Было интересно узнать много нового и про системные вызовы, и про межпроцессное взаимодействие.