Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 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 6](#_Toc123904216)

[Исходный код 7](#_Toc123904217)

[Демонстрация работы программы 10](#_Toc123904218)

[Выводы 11](#_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:

1. <iostream> - для вывода информации на консоль
2. <fstream> - для записи текста в файл.
3. <unsitd.h> - для системных вызовов read и write в Ubuntu.
4. <sys/wait.h> - для функции waitpid, когда родительский процесс ждёт дочерний.

В задании используются такие команды и строки, как:

1. **int fd[2]** - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write). И
2. **pipe(fd)** - конвейер, с помощью которого выход одной команды подается на вход другой (также “труба”). Это неименованный канал передачи данных между родственными процессами.
3. **pid\_t child = fork ()** - создание дочернего процесса, в переменной child будет храниться “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)
4. **read(int fd, void\* buf, size\_t count) (здесь дан общий пример)** - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0 (в моей программе fd1[0], fd2[0]), указатель на память получаемого объекта (переменной, массива и т.д.), размер получаемого объекта (в байтах).
5. **write(int fd, void\* buf, size\_t count) (здесь дан общий пример) -** команда, предназначенная для записи данных в другой процесс, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1 (в моей программе fd1[1], fd2[1]), указатель на память посылаемого объекта (переменной, массива и т.д.), размер посылаемогообъекта (в байтах).
6. **close(int fd)** - команда, закрывающая файловый дескриптор.
7. **int wstatus; waitpid(child, &wstatus, 0)** – команда ожидания завершения процесса с id, равным child, если она завершилась без ошибок, то в переменной wstatus будет лежать значение 0, и родительский процесс продолжит своё выполнение, в противном случае там будет лежать значение ненулевое значение, и родительский процесс аварийно завершится.

# **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 <iostream>
2. #include <fstream>
3. #include <unistd.h>
4. #include <sys/wait.h>
5. int main()
6. {
7. int fd1[2], fd2[2];
8. if ((pipe(fd1) == -1) || (pipe(fd2) == -1))
9. {
10. std::cout << "Failed to open pipes between parent and child processes" << std::endl;
11. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
12. }
13. pid\_t child;
14. if ((child = fork()) == -1)
15. {
16. std::cout << "Failed to create child process" << std::endl;
17. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
18. }
19. else if (child > 0)
20. {
21. std::cout << "You're in the parent process with id [" << getpid() << "]" << std::endl;
22. std::string file\_path, string;
23. std::cout << "Input path to the file" << std::endl;
24. getline(std::cin, file\_path);
25. int length = file\_path.length() + 1;
26. write(fd1[1], &length, sizeof(int));
27. write(fd1[1], file\_path.c\_str(), length \* sizeof(char));
28. char symbol, \*in = (char\*) [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html) (2 \* sizeof(char));
29. int counter = 0, size\_of\_in = 2;
30. std::cout << "Now input some strings. If you want to end input, press Ctrl+D" << std::endl;
31. while ((symbol = [getchar](http://www.opengroup.org/onlinepubs/009695399/functions/getchar.html)()) != EOF)
32. {
33. in[counter++] = symbol;
34. if (counter == size\_of\_in)
35. {
36. size\_of\_in \*= 2;
37. in = (char\*) [realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html) (in, size\_of\_in \* sizeof(char));
38. }
39. }
40. in = (char\*) [realloc](http://www.opengroup.org/onlinepubs/009695399/functions/realloc.html) (in, (counter + 1) \* sizeof(char));
41. in[counter] = '**\0**';
42. write(fd1[1], &(++counter), sizeof(int));
43. write(fd1[1], in, counter \* sizeof(char));
44. close(fd1[0]);
45. close(fd1[1]);
46. int out\_length, wstatus;
47. waitpid(child, &wstatus, 0);
48. if (wstatus)
49. {
50. close(fd2[0]);
51. close(fd2[1]);
52. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
53. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
54. }
55. std::cout << "You're back in parent process with id [" << getpid() << "]" << std::endl;
56. read(fd2[0], &out\_length, sizeof(int));
57. char\* out = (char\*) [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html) (out\_length);
58. read(fd2[0], out, out\_length \* sizeof(char));
59. std::ifstream fin(file\_path.c\_str());
60. if (!fin.is\_open())
61. {
62. close(fd2[0]);
63. close(fd2[1]);
64. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
65. std::cout << "Failed to open file to read strings" << std::endl;
66. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(1);
67. }
68. std::cout << "------------------------------------------------" << std::endl << "These strings end in character '.' or ';' :" << std::endl;
69. if (fin.peek() != EOF)
70. {
71. while (!fin.eof())
72. {
73. getline(fin, string);
74. std::cout << string << std::endl;
75. }
76. }
77. fin.close();
78. std::cout << "------------------------------------------------" << std::endl << "These strings don't end in character '.' or ';' :" << std::endl;
79. for (int i = 0; i < out\_length - 1; i++)
80. {
81. std::cout << out[i];
82. }
83. close(fd2[0]);
84. close(fd2[1]);
85. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(in);
86. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(out);
87. }
88. else
89. {
90. int length;
91. read(fd1[0], &length, sizeof(int));
92. char\* c\_file\_path = (char\*) [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html) (length \* sizeof(char));
93. read(fd1[0], c\_file\_path, length \* sizeof(char));
94. int counter;
95. read(fd1[0], &counter, sizeof(int));
96. char\* inc = (char\*) [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html) (counter \* sizeof(char));
97. read(fd1[0], inc, counter \* sizeof(char));
98. close(fd1[0]);
99. close(fd1[1]);
100. std::cout << "Now you are in child process with id [" << getpid() << "]" << std::endl;
101. std::string string = std::string(), out\_string = std::string(), file\_string = std::string();
102. for (int i = 0; i < counter; i++)
103. {
104. if (inc[i] != '**\0**')
105. {
106. string += inc[i];
107. }
108. if ((inc[i] == '**\n**') || (inc[i] == '**\0**'))
109. {
110. if ((i > 0) && ((inc[i - 1] == '.') || (inc[i - 1] == ';')))
111. {
112. file\_string += string;
113. }
114. else
115. {
116. out\_string += string;
117. }
118. string = std::string();
119. }
120. }
121. if ((file\_string.length()) && (file\_string[file\_string.length() - 1] == '**\n**'))
122. {
123. file\_string.pop\_back();
124. }
125. std::ofstream fout(c\_file\_path);
126. if (!fout.is\_open())
127. {
128. std::cout << "Failed to create or open file to write strings" << std::endl;
129. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(inc);
130. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(c\_file\_path);
131. close(fd2[0]);
132. close(fd2[1]);
133. return 1;
134. }
135. int out\_length = out\_string.length() + 1;
136. write(fd2[1], &out\_length, sizeof(int));
137. write(fd2[1], out\_string.c\_str(), out\_length \* sizeof(char));
138. close(fd2[0]);
139. close(fd2[1]);
140. if (!file\_string.empty())
141. {
142. fout << file\_string;
143. }
144. fout.close();
145. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(inc);
146. [free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(c\_file\_path);
147. }
148. return 0;
149. }

## **Демонстрация работы программы**



# **Выводы**

Было познавательно делать свою первую лабораторную работу по операционным системам.