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

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("You're in the parent process with id [%i]\n", getpid());
12.   char symbol, *in = (char *)malloc(sizeof(char)), *file_path = (char *)mal-
    loc(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("Mapping failed in creation of integer value\n");
17.     exit(1);
18.   }
19.   *size = 1;
20.   printf("Input path to the file\n");
21.   while ((symbol = getchar()) != '\n')
22.   {
23.     file_path[counter++] = symbol;
24.     if (counter == *size)
25.     {
26.       *size *= 2;
27.       file_path = (char *)realloc(file_path, (*size) * sizeof(char));
28.     }
29.   }
30.   file_path = (char *)realloc(file_path, (counter + 1) * sizeof(char));
31.   file_path[counter] = '\0';
32.   counter = 0, *size = 1;
33.   printf("Now input some strings. If you want to end input, press Ctrl+D\n");
34.   while ((symbol = getchar()) != EOF)
35.   {
36.     in[counter++] = symbol;
37.     if (counter == *size)
38.     {
39.       *size *= 2;
40.       in = (char *)realloc(in, (*size) * sizeof(char));
41.     }
42.   }
43.   *size = counter + 1;
44.   in = (char *)realloc(in, (*size) * sizeof(char));
45.   in[(*size) - 1] = '\0';
```

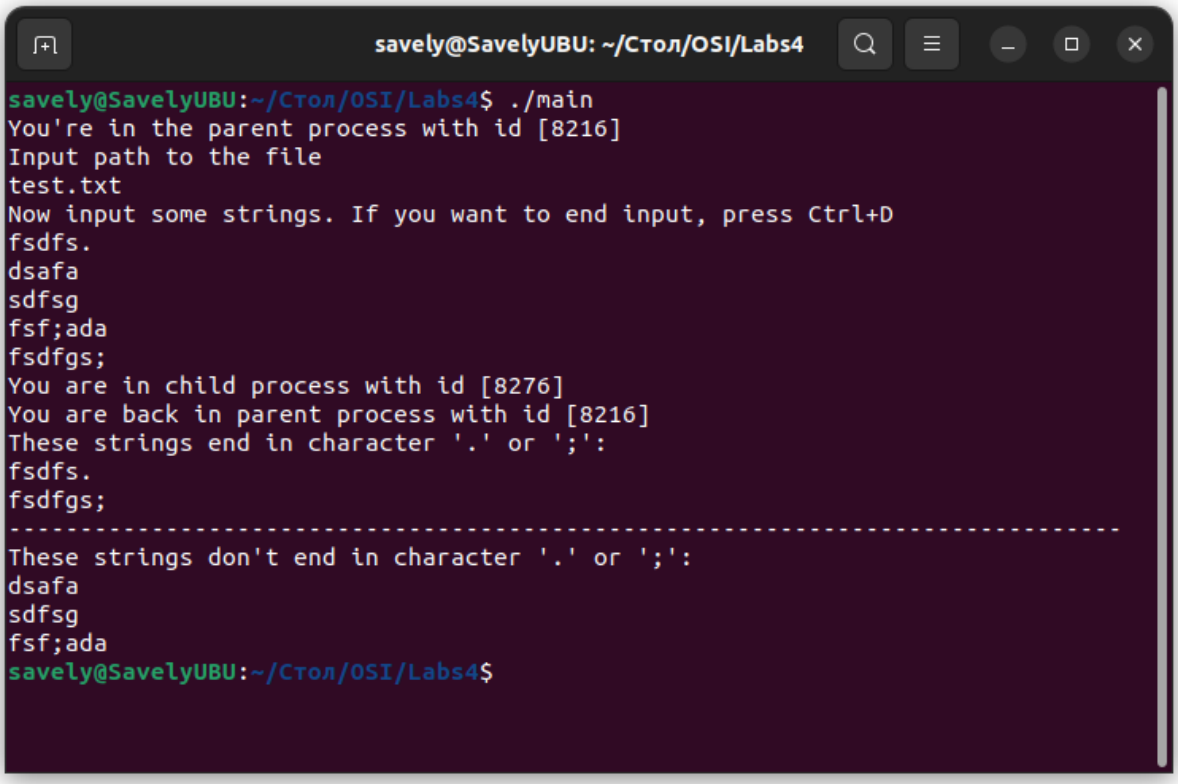
```
46.   char *ptr = (char *)mmap(NULL, (*size) * sizeof(char), PROT_READ | PROT_WRITE,
    MAP_SHARED | MAP_ANONYMOUS, 0, 0);
47.   if (ptr == MAP_FAILED)
48.   {
49.     printf("Mapping failed in creation of array of chars\n");
50.     free(in);
51.     free(file_path);
52.     int err = munmap(size, sizeof(int));
53.     if (err != 0)
54.     {
55.       printf("Unmapping failed\n");
56.     }
57.     exit(1);
58.   }
59.   strcpy(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("Failed to open file\n");
64.     free(in);
65.     free(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("Unmapping failed\n");
71.     }
72.     exit(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("Failed to create string associated with file\n");
78.     free(in);
79.     free(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("Unmapping failed\n");
85.     }
86.     exit(1);
87.   }
88.   pid_t child_pid = fork();
89.   if (child_pid == -1)
90.   {
91.     printf("Failed to create child process\n");
92.     free(in);
93.     free(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("Unmapping failed\n");
99.     }
100.          exit(1);
101.        }
102.        else if (child_pid == 0)
103.        {
104.          //child
105.          printf("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("Failed to truncate file\n");
135.              free(in);
136.              free(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("Failed to resize memory for string associated with file\n");
```

```
142.               free(in);
143.               free(file_path);
144.               return 1;
145.             }
146.             sprintf(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("Failed to truncate file for string\n");
155.             free(in);
156.             free(file_path);
157.             return 1;
158.           }
159.           *size = out_string.length() + 1;
160.           sprintf(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(in);
170.             free(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("Unmapping failed\n");
177.             }
178.             exit(1);
179.           }
180.           printf("You are back in parent process with id [%i]\n", getpid());
181.           struct stat statbuf;
182.           if (fstat(fd, &statbuf) < 0)
183.           {
184.             printf("Problems with opening file %s\n", file_path);
185.             exit(1);
186.           }
187.           counter = std::max((int)statbuf.st_size, 1);
188.           printf("These strings end in character '.' or ';':\n");
189.           if (statbuf.st_size > 1)
190.           {
191.             printf("%s", f);
```

```
192.          }
193.          printf("----------------------------------------------------------------------
    ----------\nThese strings don't end in character '.' or ';':\n");
194.          printf("%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("Unmapping failed\n");
202.            free(in);
203.            free(file_path);
204.            exit(1);
205.          }
206.        }
207.        free(in);
208.        free(file_path);
209.        return 0;
210.      }
```

Демонстрация работы программы



Выводы

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