Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Лабораторная работа №4 по курсу «Операционные системы»

Освоение принципов работы с файловыми системами. Обеспечение обмена данных между процессами посредством технологии «File mapping».

Студент: П. Ф. Гришин
Преподаватель: Е. С. Миронов
Группа: М8О-201Б-21
Вариант: 16
Дата:
Оценка:
Подпись:

Москва, 2023

1 Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в резуль- тате работы. Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс долж- ны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода. Правило проверки: строка должна оканчиваться на «.» или «;»

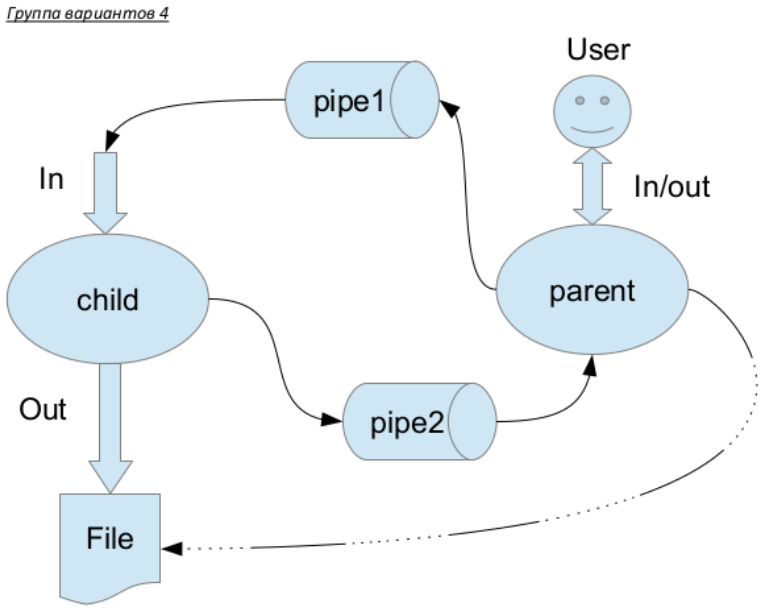

Рис. 1: Структура программы

2 Сведения о программе

Программа написанна на Си в Unix подобной операционной системе на базе ядра Linux. В программе создается дочерний процесс, в который перенаправляются данные из pipe.

Дочерний прочесс принимает строку чисел и находит их сумму, ответ записывая в файл. Имя файла задается пользователем Родительский процесс считывает вводные данные у пользовотеля и пердет их дочер- нему процессу через pipe.

Программа завершает работу при окончании ввода, то есть нажатии CTRL+D.

3 Общий метод и алгоритм решения

Все проверки совершаются через функцию Handle_error.

При запуске программы прользоваательльь может ввести имя файла, который создаст дочерняя программа. После запуска создается pipe, два файловых дескриптора для потока ошибок и для общения между родительским и дочерним процессом.

Родительский процесс просит пользователя ввести имя файла, в котором содержатся имя выходного файла и данные. После проверки файла на ошибки начинается передача через файловый дескриптор src_fd строки. После этого создается дочерний процесс с помощью **fork()**. В нем дескриптор потока ввода заменяется на поток вывода из pipe. Таким образом, когда родитель запишет что-то в pipe ребенок сможет это считать, как будто ввод происходит из консоли. После проверки файла на ошибки начинается передача через файловый дескриптор src_fd строки. После замены дескрипторов вызывается дочерняя прогграмма с помощью **execl()**. В нее имя файла передается как параметр при запуске. Далее дочерний процесс обработает строку и выведет свой вердикт подходит ли данная строка или нет. Если она подходит, то дочерний процесс даст сигнал родителю, что строка валидна. Иначе - вызовет поток ошибок (файловый дескриптор error_fd) и передаст родительскому процессу. В свою очередь, родитель выведет пользователю сообщение о том, что строка не подходит.

При нажатии CTRL+D пользователь сигнализирует о конце ввода. Родительский процесс завершает работу, а вместе с ним и дочерний.

4 Листинг программы

parent.c

```
1 #include "library.h"
2 #include "utils.h"
3
```

```
4  void ParentRoutine(char* nameF){
5     FILE* file = fopen(nameF, "r");
6      Handle_error(file == NULL , "open error");
7
8     int src_fd[2];
9     int pipe_response = pipe(src_fd);
10    Handle_error(pipe_response == -1, "pipe error");
11
12    int err_fd[2];
13    pipe_response = pipe(err_fd);
14    Handle_error(pipe_response == -1, "pipe error");
15
16    pid_t id = fork();
17    Handle_error(id == -1, "fork error");
18
19    if (id == 0){
20
21        char name[64];
22        read(src_fd[0], &name, sizeof(name));
23
24        char *src_fd_0, *src_fd_1, *err_fd_0, *err_fd_1;
25        asprintf(&src_fd_0, "%d", src_fd[0]);
26        asprintf(&src_fd_1, "%d", src_fd[1]);
27        asprintf(&err_fd_0, "%d", err_fd[0]);
28        asprintf(&err_fd_1, "%d", err_fd[1]);
29
30        execl("child", name, src_fd_0, src_fd_1, err_fd_0, err_fd_1, NULL);
31
32    } else {
33        char* parent;
34        int parent_pid = getpid();
35        printf("[%d] PARENT. ",parent_pid);
36        printf("Enter the name of file to write: ");
37        char name[256];
38        fscanf(file, "%s", name);
39        Clean(name);
40        printf("%s\n", name);
41        write(src_fd[1], &name, sizeof(name));
42        bool file_error;
43        read(err_fd[0], &file_error, sizeof(bool));
44        if (file_error){
45            close(src_fd[0]); close(src_fd[1]);
46            close(err_fd[0]); close(err_fd[1]);
47            Handle_error(true, "file error\n");
48        }
49
50        char str[256];
51        printf("[%d] PARENT. ",parent_pid);
52        printf("Enter string: ");
```

```
53          while (fscanf(file, "%s", str) != EOF){
54              Clean(str);
55              printf("%s\n", str);
56              write(src_fd[1], &str, sizeof(str));
57              bool err;
58              read(err_fd[0], &err, sizeof(bool));
59              if (err){
60                  char* err_msg;
61                  asprintf(&err_msg, "Error: \"%s\" is not valid.\n", str);
62                  printf("[%d] PARENT. ",parent_pid);
63                  printf("Error: \"%s\" is not valid.\n", str);
64              }
65              printf("[%d] PARENT. ",parent_pid);
66              printf("Enter string: ");
67          }
68          write(src_fd[1], "_quit", sizeof(str));
69
70      }
71      write(fileno(stdout), "\n", sizeof "\n");
72      close(src_fd[0]); close(src_fd[1]);
73      close(err_fd[0]); close(err_fd[1]);
74      fclose(file);
75  }
```

child.c

```
1   #include "library.h"
2   #include "utils.h"
3
4   int main(int argv, char* argc[]){
5
6       int src_fd[2], err_fd[2];
7       src_fd[0] = atoi(argc[1]);
8       src_fd[1] = atoi(argc[2]);
9       err_fd[0] = atoi(argc[3]);
10      err_fd[1] = atoi(argc[4]);
11
12      char* name = argc[0];
13      int output_fd = open(name, O_WRONLY | O_CREAT);
14      bool file_error = false;
15      if (output_fd < 0) file_error = true;
16      write(err_fd[1], &file_error, sizeof(bool));
17      if (file_error){
18          close(src_fd[0]); close(src_fd[1]);
19          close(err_fd[0]); close(err_fd[1]);
20      }
21
22      char str[256];
23      read(src_fd[0], &str, sizeof(str));
24      read(src_fd[0], &str, sizeof(str));
```

```
25      while(strcmp(str, "_quit") != 0){
26          bool err;
27          int lastIndex = StrLength(str);
28          if (StrLength(str)){
29              err = false;
30              write(output_fd, str, (strlen(str)) * sizeof(char));
31              write(output_fd, "\n", 1);
32          } else {
33              err = true;
34          }
35          write(err_fd[1], &err, sizeof(bool));
36          read(src_fd[0], &str, sizeof(str));
37      }
38      close(src_fd[0]); close(src_fd[1]);
39      close(err_fd[0]); close(err_fd[1]);
40  }
```

5 Демонстрация работы программы

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/OS/lab2$ ./Lab2
test.txt
[43971] PARENT. Enter the name of file to write: OutputFile.txt
[43971] PARENT. Enter string: FirstTest.
[43971] PARENT. Enter string: False
[43971] PARENT. Error: "False" is not valid.
[43971] PARENT. Enter string: CCCCCCCCCCCCCCCCCCC.
[43971] PARENT. Enter string: End;
[43971] PARENT. Enter string: ........................a
[43971] PARENT. Error: "........................a" is not valid.
[43971] PARENT. Enter string: wave...;
[43971] PARENT. Enter string: wrongString
[43971] PARENT. Error: "wrongString" is not valid.

[43971] PARENT. Enter string:
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/OS/lab4$
```

6 Вывод

Одна из основных задач операционной системы - это управление процессами. В большинстве случаев она сама создает процессы для себя и при запуске других программ. Тем не менее бывают случаи, когда необходимо создавать процессы вручную.

В языке Си есть функционал, который позволит нам внутри нашей программы создать дополнительный, дочерний процесс. Этот процесс будет работать параллельно с родительским.

Для создания дочерних процессов используется функция fork. При этом с помощью ветвлений в коде можно отделить код родителя от ребенка. У ребенка при этом можно заменить программу, испрользуя для этого функцию exec, а обеспечить связь с помощью pipe.

Подобный функционал есть во многих языках программирования, так как большинство современных программ состоят более, чем из одного процесса.