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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс передает команды пользователя дочернему процессу. Дочерний процесс принеобходимости передает данные в родительский процесс. Результаты своей работы дочерний процесс пишет в созданный им файл.

Правило проверки: строка должна оканчиваться на «.» или «;»

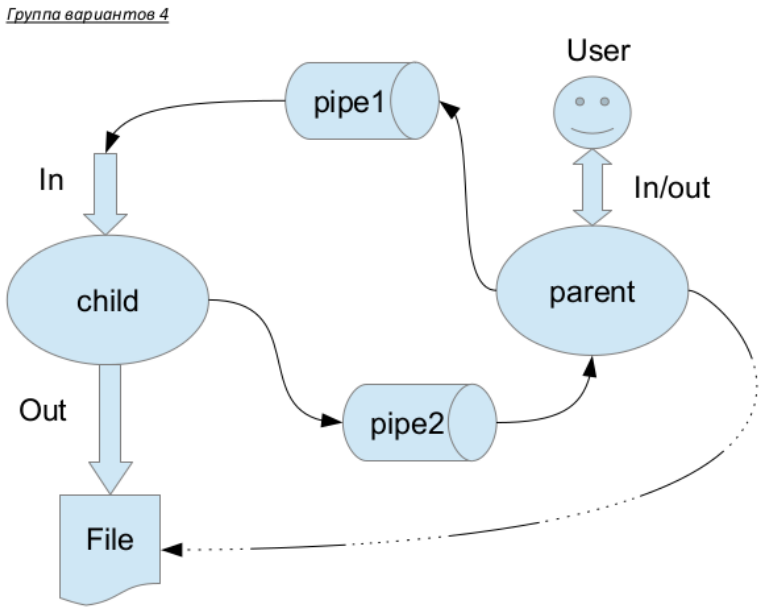

Рис. 1: Структура программы

2 Сведения о программе

Программа написанна на Си в Unix подобной операционной системе на базе ядра Linux. При компиляции следует линковать библиотеки -lpthread и -lrt. В програм-

ме создается дочерний процесс, данные в который передаются с помощью shared memory.

Дочерний прочесс принимает строку и проверяет подходит ли данная строка в соответствии с правилом, ответ записывая в файл. Имя файла задается пользователем.

Родительский процесс считывает вводные данные у пользовотеля и пердет их дочернему процессу через отброженный участок памати shared memory.

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

3 Листинг программы

pc.c

```
#include "library.h"
void handle_error(bool expr, char* msg){
    if (expr){
        perror(msg);
        exit(-1);
    }
}

void clean(char* str){
    for (int i = 0 ; i < strlen(str); ++i){
        if (str[i] == '\n'){ str[i] = '\0'; }
    }
}

bool StrLength(char* str) {
    return str[strlen(str) - 1] == '.' || str[strlen(str) - 1] == ';';
}

int ParentRoutine(char* nameF){
    FILE* file = fopen(nameF, "r");
    handle_error(file == NULL , "open error");

    const char* SOURCE_SEMAPHORE_NAME = "source_sem";
    const char* RESPONSE_SEMAPHORE_NAME = "response_sem";

    sem_unlink(SOURCE_SEMAPHORE_NAME);
    sem_unlink(RESPONSE_SEMAPHORE_NAME);

    const char* SOURCE_NAME = "source_shm";
    const char* RESPONSE_NAME = "response_shm";
    const int SIZE = 4096;

    shm_unlink(SOURCE_NAME);
    shm_unlink(RESPONSE_NAME);
```

```
35
36      int source_fd = shm_open(SOURCE_NAME, O_RDWR | O_CREAT, 0644);
37      int response_fd = shm_open(RESPONSE_NAME, O_RDWR | O_CREAT, 0644);
38      handle_error(source_fd == -1 || response_fd == -1, "can't open shared memory object
            ");
39
40      handle_error(ftruncate(source_fd, SIZE) == -1 ||
41                   ftruncate(response_fd, SIZE) == -1,
42                   "can't resize shared memory object");
43
44      void* source_ptr = mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, source_fd,
            0);
45      void* response_ptr = mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
            response_fd, 0);
46      handle_error(source_ptr == MAP_FAILED || response_ptr == MAP_FAILED, "can't mmap
            shared memory object");
47
48      sem_t* source_semaphore = sem_open(SOURCE_SEMAPHORE_NAME, O_RDWR | O_CREAT, 0644,
            0);
49      sem_t* response_semaphore = sem_open(RESPONSE_SEMAPHORE_NAME, O_RDWR | O_CREAT,
            0644, 0);
50      handle_error(source_semaphore == SEM_FAILED ||
51                   response_semaphore == SEM_FAILED,
52                   "can't open semaphore");
53
54      pid_t pid = fork();
55      handle_error(pid == -1, "fork error");
56
57      if (pid == 0){
58
59          sem_wait(source_semaphore);
60          char* name = source_ptr;
61          int output_fd = open(name, O_WRONLY | O_CREAT, 0644);
62          char* file_error = "0";
63          if (output_fd < 0){ file_error = "1"; }
64          strcpy(response_ptr, file_error);
65          sem_post(response_semaphore);
66
67          sem_wait(source_semaphore);
68          char* str = source_ptr;
69          while (strcmp(str, "\0") != 0){
70              char* error = "0";
71              if (StrLength(str)){
72                  write(output_fd, str, strlen(str) * sizeof(char));
73                  write(output_fd, "\n", sizeof "\n");
74              } else {
75                  error = "1";
76              }
77              strcpy(response_ptr, error);
```

```
78              sem_post(response_semaphore);
79              sem_wait(source_semaphore);
80              str = source_ptr;
81          }
82
83      } else {
84
85          const int parent = getpid();
86          printf("[%d] Enter the name of file to write: ", parent);
87          fflush(stdout);
88          char name[256];
89          fscanf(file, "%s", name);
90          clean(name);
91          printf("%s\n", name);
92          strcpy(source_ptr, name);
93          sem_post(source_semaphore);
94
95          sem_wait(response_semaphore);
96          if (strcmp(response_ptr, "1") == 0){
97              close(source_fd);
98              close(response_fd);
99              shm_unlink(SOURCE_NAME);
100             shm_unlink(RESPONSE_NAME);
101             munmap(source_ptr, SIZE);
102             munmap(response_ptr, SIZE);
103             sem_destroy(source_semaphore);
104             sem_destroy(response_semaphore);
105             sem_unlink(SOURCE_SEMAPHORE_NAME);
106             sem_unlink(RESPONSE_SEMAPHORE_NAME);
107             handle_error(true, "file error");
108         }
109
110         char str[256];
111         printf("[%d] Enter string: ", parent);
112         fflush(stdout);
113         while (fscanf(file, "%s", str) != EOF){
114             printf("%s\n", str);
115             clean(str);
116             strcpy(source_ptr, str);
117             sem_post(source_semaphore);
118             sem_wait(response_semaphore);
119             char* error = response_ptr;
120             if (strcmp(error, "1") == 0){
121                 printf("Error: \"%s\" is not valid.\n", str);
122             }
123             printf("[%d] Enter string: ", parent);
124             fflush(stdout);
125         }
126
```

```
127         printf("\n");
128         fflush(stdout);
129 
130     }
131 
132     close(source_fd);
133     close(response_fd);
134     shm_unlink(SOURCE_NAME);
135     shm_unlink(RESPONSE_NAME);
136     munmap(source_ptr, SIZE);
137     munmap(response_ptr, SIZE);
138     sem_destroy(source_semaphore);
139     sem_destroy(response_semaphore);
140     sem_unlink(SOURCE_SEMAPHORE_NAME);
141     sem_unlink(RESPONSE_SEMAPHORE_NAME);
142     fclose(file);
143 }
```

4 Демонстрация работы программы

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/OS/lab4$ ./Lab4
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$
```

5 Вывод

Взаимодействие между процессами можно организовать при помощи каналов, сокетов и отображаемых файлов. В данной лабораторной работе был изучен и применен механизм межпроцессорного взаимодействия – file mapping. Файл отображается на оперативную память таким образом, что мы можем взаимодействовать с ним как с массивом.

Благодаря этому вместо медленных запросов на чтение и запись мы выполняем отображение файла в ОЗУ и получаем произвольный доступ за O(1). Из-за этого при использовании этой технологии межпроцессорного взаимодействия мы можем получить ускорении работы программы, в сравнении, с использованием каналов.

Из недостатков данного метода можно выделить то, что дочерние процессы обязательно должны знать имя отображаемого файла и также самостоятельно выполнить отображение.