МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Институт №8 "Информационные технологий и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №4
по курсу "Операционные системы"
3 семестр

Студент: Леухин М. В.
Группа: М8О-206Б-20
Преподаватель: Соколов А. А.
Дата: 20.11.21
Оценка: 5
Подпись: _____

Москва, 2021

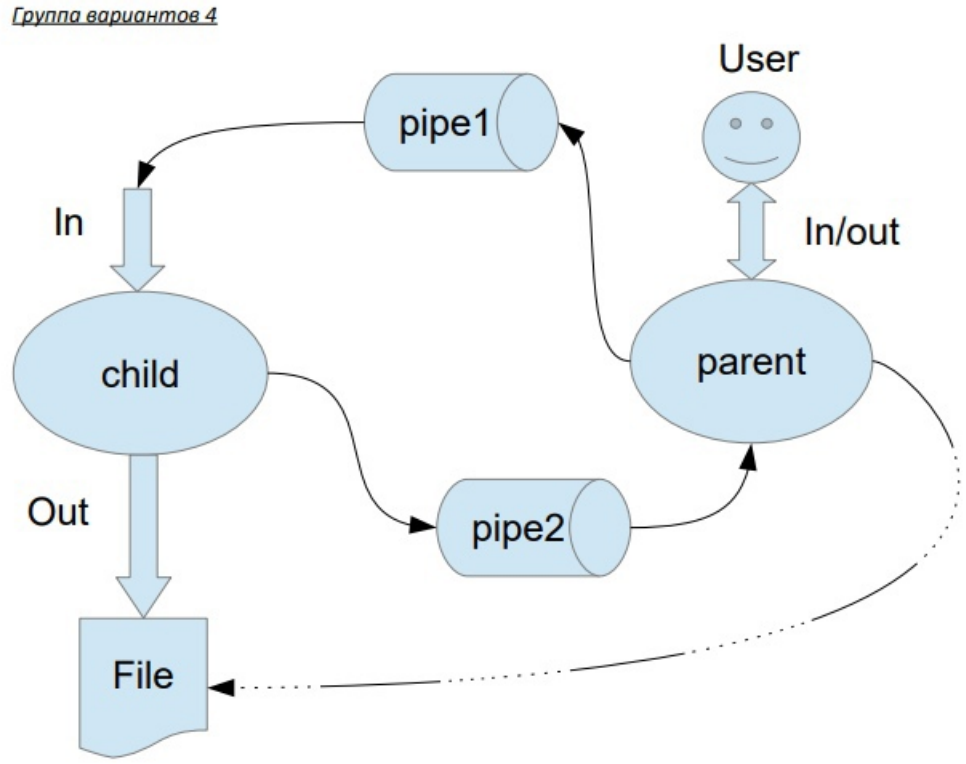
Содержание

1 Постановка задачи

Цель работы: приобретение практических навыков в управлении процессами в ОС и обеспечении обмена данных между процессами посредством технологии "File mapping".

Задание: 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 через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 15: 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 должны 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 начинаться с заглавной буквы.



2 Основная часть

2.1 Листинг программы

Файл main.c:

```c
#define _GNU_SOURCE

#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <string.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'; }
    }
}

int main(){

    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);

    int source_fd = shm_open(SOURCE_NAME, O_RDWR | O_CREAT, 0644);
    int response_fd = shm_open(RESPONSE_NAME, O_RDWR | O_CREAT,
        0644);
```

```
43      handle_error(source_fd == -1 || response_fd == -1, "can't
            open shared memory object");
44
45      handle_error(ftruncate(source_fd, SIZE) == -1 ||
46                   ftruncate(response_fd, SIZE) == -1,
47                   "can't resize shared memory object");
48
49      void* source_ptr = mmap(NULL, SIZE, PROT_READ | PROT_WRITE,
            MAP_SHARED, source_fd, 0);
50      void* response_ptr = mmap(NULL, SIZE, PROT_READ | PROT_WRITE,
            MAP_SHARED, response_fd, 0);
51      handle_error(source_ptr == MAP_FAILED || response_ptr ==
            MAP_FAILED, "can't mmap shared memory object");
52
53      sem_t* source_semaphore = sem_open(SOURCE_SEMAPHORE_NAME,
            O_RDWR | O_CREAT, 0644, 0);
54      sem_t* response_semaphore = sem_open(RESPONSE_SEMAPHORE_NAME,
            O_RDWR | O_CREAT, 0644, 0);
55      handle_error(source_semaphore == SEM_FAILED ||
56                   response_semaphore == SEM_FAILED,
57                   "can't open semaphore");
58
59      pid_t pid = fork();
60      handle_error(pid == -1, "fork error");
61
62      if (pid == 0){
63
64          sem_wait(source_semaphore);
65          char* name = source_ptr;
66          int output_fd = open(name, O_WRONLY | O_CREAT, 0644);
67          char* file_error = "0";
68          if (output_fd < 0){ file_error = "1"; }
69          strcpy(response_ptr, file_error);
70          sem_post(response_semaphore);
71
72          sem_wait(source_semaphore);
73          char* str = source_ptr;
74          while (strcmp(str, "\0") != 0){
75              char* error = "0";
76              if (str[0] >= 'A' && str[0] <= 'Z'){
77                  write(output_fd, str, strlen(str) * sizeof(char));
78                  write(output_fd, "\n", sizeof "\n");
79              } else {
80                  error = "1";
81              }
82              strcpy(response_ptr, error);
83              sem_post(response_semaphore);
84              sem_wait(source_semaphore);
85              str = source_ptr;
86          }
```

```
87
88      } else {
89
90          const int parent = getpid();
91          printf("[%d] Enter the name of file to write: ", parent);
92          fflush(stdout);
93          char name[256];
94          read(fileno(stdin), name, 256); clean(name);
95          strcpy(source_ptr, name);
96          sem_post(source_semaphore);
97
98          sem_wait(response_semaphore);
99          if (strcmp(response_ptr, "1") == 0){
100             close(source_fd);
101             close(response_fd);
102             shm_unlink(SOURCE_NAME);
103             shm_unlink(RESPONSE_NAME);
104             munmap(source_ptr, SIZE);
105             munmap(response_ptr, SIZE);
106             sem_destroy(source_semaphore);
107             sem_destroy(response_semaphore);
108             sem_unlink(SOURCE_SEMAPHORE_NAME);
109             sem_unlink(RESPONSE_SEMAPHORE_NAME);
110             handle_error(true, "file error");
111         }
112
113         char str[256];
114         printf("[%d] Enter string: ", parent);
115         fflush(stdout);
116         while (read(fileno(stdin), str, 256) != 0){
117             clean(str);
118             strcpy(source_ptr, str);
119             sem_post(source_semaphore);
120             sem_wait(response_semaphore);
121             char* error = response_ptr;
122             if (strcmp(error, "1") == 0){
123                 printf("Error: \"%s\" is not valid.\n", str);
124             }
125             printf("[%d] Enter string: ", parent);
126             fflush(stdout);
127         }
128
129         printf("\n");
130         fflush(stdout);
131
132     }
133
134     close(source_fd);
135     close(response_fd);
136     shm_unlink(SOURCE_NAME);
```

```
137         shm_unlink(RESPONSE_NAME);
138         munmap(source_ptr, SIZE);
139         munmap(response_ptr, SIZE);
140         sem_destroy(source_semaphore);
141         sem_destroy(response_semaphore);
142         sem_unlink(SOURCE_SEMAPHORE_NAME);
143         sem_unlink(RESPONSE_SEMAPHORE_NAME);
144 
145 }
```

2.2 Результат работы программы

```
1 matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/2os/4lab/src$
    ./a.out
2 [24647] PARENT. Enter the name of file to write: output.txt
3 [24647] PARENT. Enter string: This is valid string
4 [24647] PARENT. Enter string: invalid str
5 [24647] PARENT. Error: "invalid str" is not valid.
6 [24647] PARENT. Enter string:
7 matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/2os/4lab/src$ cat
    output.txt
8 This is valid string
```

3 Вывод

В ходе выполнения лабораторной работы я познакомился с технологией "File mapping", а также с инструментами, которые операционная система предоставляет для релаизации этой технологии. Самое интересное в этой лабораторной работе было организовать синхронизацию между процессами - для этого мне пришлось изучить то, какие бывают средства синхронизации и как их использовать.