Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-212БВ-24

Студент: Головин В. П.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.10.25

Москва, 2025

Постановка задачи

Вариант 18.

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 результате работы.

Вариант 18) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создание неименованного канала для передачи данных между процессами
- int execl(const char **path*, const char **arg*, ...); - замена образа памяти процесса
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрыть файл
- void exit(int status) - завершения выполнения процесса и возвращение статуса

Parent:

- Чтение имен файлов из стандартного ввода

- Создание pipe`ов

- Создание двух процессов

- Закрытие неиспользуемых каналов

- Перенаправляем stdin на pipe[0] каждого дочернего процессами

- Чтение строк и перенаправление их к дочерним процессам с учетом правила фильтрации

- Завершение ввода (EOF)

- Ожидание завершения всех дочерних процессов

Child:

- Получение имени файла из Parent

- Чтение данных из перенаправленного stdin

- Удаление всех гласных букв

- Запись результата в файл

- Завершение процесса

Код программы

parent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>

#define MAX_LENGTH 512
#define NUM_PROCESSES 2
#define BUFFER_SIZE 4096

pid_t create_process() {
pid_t pid = fork();
if (pid == -1) {
perror("error: failed to spawn new process\n");
exit(EXIT_FAILURE);
}
return pid;
}

int create_pipe(int pipe_fd[2]) {
if (pipe(pipe_fd) == -1) {
perror("error: failed to create pipe\n");
exit(EXIT_FAILURE);
}
return 0;
}

int main() {
char file_names[NUM_PROCESSES][MAX_LENGTH];
char buffer[BUFFER_SIZE];
ssize_t bytes_read;
```

```
for (int i = 0; i < NUM_PROCESSES; i++) {
bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1);
if (bytes_read <= 0) {
perror("error: failed to read filename\n");
exit(EXIT_FAILURE);
}
buffer[bytes_read] = '\0';
char *newline = strchr(buffer, '\n');
if (newline) {
*newline = '\0';
strncpy(file_names[i], buffer, MAX_LENGTH - 1);
} else {
strncpy(file_names[i], buffer, MAX_LENGTH - 1);
}
file_names[i][MAX_LENGTH - 1] = '\0';
}

int pipe_fds[NUM_PROCESSES][2];
for (int i = 0; i < NUM_PROCESSES; i++) {
create_pipe(pipe_fds[i]);
}

pid_t pids[NUM_PROCESSES];
for (int i = 0; i < NUM_PROCESSES; i++) {
pids[i] = create_process();
if (pids[i] == 0) {
close(pipe_fds[i][1]);
dup2(pipe_fds[i][0], STDIN_FILENO);
close(pipe_fds[i][0]);
for (int j = 0; j < NUM_PROCESSES; j++) {
if (j != i) {
close(pipe_fds[j][0]);
close(pipe_fds[j][1]);
}
}
execl("./child", "child", file_names[i], NULL);
```

```
perror("error: exec failed\n");
exit(EXIT_FAILURE);
}
}

for (int i = 0; i < NUM_PROCESSES; i++) {
close(pipe_fds[i][0]);
}

int line_counter = 0;
while ((bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {
char *current_pos = buffer;
ssize_t remaining = bytes_read;
while (remaining > 0) {
char *newline = memchr(current_pos, '\n', remaining);
ssize_t chunk_size;
if (newline) {
chunk_size = newline - current_pos + 1;
} else {
chunk_size = remaining;
}
line_counter++;
int target_process = (line_counter % 2 == 1) ? 0 : 1;
write(pipe_fds[target_process][1], current_pos, chunk_size);
current_pos += chunk_size;
remaining -= chunk_size;
}
}

if (bytes_read < 0) {
perror("error: failed to read from stdin\n");
exit(EXIT_FAILURE);
}

const char eof[] = {1, '\n'};
for (int i = 0; i < NUM_PROCESSES; i++) {
```

```
write(pipe_fds[i][1], eof, sizeof(eof));
close(pipe_fds[i][1]);
}

wait(NULL);
wait(NULL);
}
```

child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>

#define MAX_LENGTH 512
#define BUFFER_SIZE 4096

int main(int argc, char* argv[]) {
if (argc != 2) {
exit(EXIT_FAILURE);
}

char* filename = argv[1];
int output_fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0600);
if (output_fd == -1) {
perror("error: can`t open file");
exit(EXIT_FAILURE);
}

dup2(output_fd, STDOUT_FILENO);
close(output_fd);

const char vowels[] = {'a', 'e', 'i', 'o', 'u', 'y',
'A', 'E', 'I', 'O', 'U', 'Y'};
```

```
const int vowels_count = sizeof(vowels) / sizeof(vowels[0]);
char input_buffer[BUFFER_SIZE];
char output_buffer[BUFFER_SIZE];
ssize_t bytes_read;
while ((bytes_read = read(STDIN_FILENO, input_buffer, BUFFER_SIZE)) > 0) {
if (bytes_read >= 1 && input_buffer[0] == 1) {
break;
}
int output_index = 0;
for (int i = 0; i < bytes_read; i++) {
if (memchr(vowels, input_buffer[i], vowels_count) == NULL) {
output_buffer[output_index++] = input_buffer[i];
}
}
if (output_index > 0) {
write(STDOUT_FILENO, output_buffer, output_index);
}
}
if (bytes_read < 0) {
perror("error: read failed in child");
exit(EXIT_FAILURE);
}
return 0;
}
```

Протокол работы программы

```
spr0vay@spr0vayhost:~/Документы/Coding/Labs/OS_Labs/Lab1$ ./parent
child1.txt
child2.txt
HellO World
It`s me MarriO
This is a test
Enough
spr0vay@spr0vayhost:~/Документы/Coding/Labs/OS_Labs/Lab1$ cat child1.txt
Hll Wrld
Ths s  tst
spr0vay@spr0vayhost:~/Документы/Coding/Labs/OS_Labs/Lab1$ cat child2.txt
```

t`s m Mrrr
ngh

Вывод

В процессе выполнения лабораторной работы я составил программу, осуществляющую работу с процессами и взаимодействие между ними. Я приобрел базовые практические навыки в управлении процессами в ОС и обеспечении обмена между процессами посредством каналов. Одной из основных сложностей стало чтение через read, а не через fgets.