Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-212БВ-24

Студент: Головин В. П.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.11.25

Москва, 2025

Постановка задачи

Переписать код первой лабораторной работы, с использованием Shared Memory и семафоров.

Общий метод и алгоритм решения

Server:

- Чтение имен файлов из стандартного ввода

- Создание Shared Memory и семафор

- Создание двух процессов

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

- Завершение ввода (EOF)

- Ожидание завершения всех дочерних процессов

Client:

- Получение имени файла из Server

- Чтение данных из перенаправленного stdin

- Удаление всех гласных букв

- Запись результата в файл

- Завершение процесса

Код программы

server.c

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

#define MAX_LENGTH 512
#define NUM_PROCESSES 2
#define BUFFER_SIZE 4096
#define SHM_SIZE (BUFFER_SIZE + sizeof(uint32_t))

typedef struct {
uint32_t length;
char data[BUFFER_SIZE];
```

```
} shm_data_t;


void generate_names(char* shm_name, char* sem_name, int index, size_t size) {
pid_t pid = getpid();
snprintf(shm_name, size, "lab3-shm-%d-%d", pid, index);
snprintf(sem_name, size, "lab3-sem-%d-%d", pid, index);
}

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

char shm_names[NUM_PROCESSES][64];
char sem_names[NUM_PROCESSES][64];
int shm_fds[NUM_PROCESSES];
sem_t *sems[NUM_PROCESSES];
shm_data_t *shm_data[NUM_PROCESSES];

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';
}

for (int i = 0; i < NUM_PROCESSES; i++) {
generate_names(shm_names[i], sem_names[i], i, sizeof(shm_names[i]));

shm_fds[i] = shm_open(shm_names[i], O_RDWR | O_CREAT | O_EXCL, 0600);
if (shm_fds[i] == -1) {
perror("error: failed to create SHM\n");
for (int j = 0; j < i; j++) {
```

```
close(shm_fds[j]);
shm_unlink(shm_names[j]);
sem_close(sems[j]);
sem_unlink(sem_names[j]);
}
exit(EXIT_FAILURE);
}

if (ftruncate(shm_fds[i], SHM_SIZE) == -1) {
perror("error: failed to set SHM size\n");
close(shm_fds[i]);
shm_unlink(shm_names[i]);
for (int j = 0; j < i; j++) {
close(shm_fds[j]);
shm_unlink(shm_names[j]);
sem_close(sems[j]);
sem_unlink(sem_names[j]);
}
exit(EXIT_FAILURE);
}

shm_data[i] = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fds[i], 0);
if (shm_data[i] == MAP_FAILED) {
perror("error: failed to map SHM\n");
close(shm_fds[i]);
shm_unlink(shm_names[i]);
for (int j = 0; j < i; j++) {
close(shm_fds[j]);
shm_unlink(shm_names[j]);
sem_close(sems[j]);
sem_unlink(sem_names[j]);
}
exit(EXIT_FAILURE);
}

shm_data[i]->length = 0;

sems[i] = sem_open(sem_names[i], O_CREAT | O_EXCL, 0600, 1);
if (sems[i] == SEM_FAILED) {
perror("error: failed to create semaphore\n");
munmap(shm_data[i], SHM_SIZE);
close(shm_fds[i]);
shm_unlink(shm_names[i]);
for (int j = 0; j < i; j++) {
munmap(shm_data[j], SHM_SIZE);
close(shm_fds[j]);
shm_unlink(shm_names[j]);
sem_close(sems[j]);
sem_unlink(sem_names[j]);
}
exit(EXIT_FAILURE);
}
}
```

```
pid_t pids[NUM_PROCESSES];
for (int i = 0; i < NUM_PROCESSES; i++) {
pids[i] = create_process();
if (pids[i] == 0) {
execl("./client", "client", file_names[i], shm_names[i], sem_names[i], NULL);
perror("error: exec failed\n");
exit(EXIT_FAILURE);
}
}

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;
sem_wait(sems[target_process]);
if (chunk_size <= BUFFER_SIZE) {
shm_data[target_process]->length = chunk_size;
memcpy(shm_data[target_process]->data, current_pos, chunk_size);
}
sem_post(sems[target_process]);
current_pos += chunk_size;
remaining -= chunk_size;
}
}

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

for (int i = 0; i < NUM_PROCESSES; i++) {
sem_wait(sems[i]);
shm_data[i]->length = UINT32_MAX;
sem_post(sems[i]);
}

for (int i = 0; i < NUM_PROCESSES; i++) {
waitpid(pids[i], NULL, 0);
}

for (int i = 0; i < NUM_PROCESSES; i++) {
munmap(shm_data[i], SHM_SIZE);
close(shm_fds[i]);
```

```
shm_unlink(shm_names[i]);
sem_close(sems[i]);
sem_unlink(sem_names[i]);
}

return 0;
}
```

cient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <stdint.h>
#include <stdbool.h>

#define BUFFER_SIZE 4096
#define SHM_SIZE (BUFFER_SIZE + sizeof(uint32_t))

typedef struct {
uint32_t length;
char data[BUFFER_SIZE];
} shm_data_t;

int main(int argc, char* argv[]) {
if (argc != 4) {
exit(EXIT_FAILURE);
}

char* filename = argv[1];
char* shm_name = argv[2];
char* sem_name = argv[3];
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);
}

int shm_fd = shm_open(shm_name, O_RDWR, 0);
if (shm_fd == -1) {
perror("error: failed to open SHM");
close(output_fd);
exit(EXIT_FAILURE);
}

shm_data_t *shm_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
if (shm_data == MAP_FAILED) {
perror("error: failed to map SHM");
close(shm_fd);
```

```
close(output_fd);
exit(EXIT_FAILURE);
}

sem_t *sem = sem_open(sem_name, O_RDWR);
if (sem == SEM_FAILED) {
perror("error: failed to open semaphore");
munmap(shm_data, SHM_SIZE);
close(shm_fd);
close(output_fd);
exit(EXIT_FAILURE);
}

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 output_buffer[BUFFER_SIZE];
bool running = true;
while (running) {
sem_wait(sem);
if (shm_data->length == UINT32_MAX) {
running = false;
} else if (shm_data->length > 0) {
int output_index = 0;
for (uint32_t i = 0; i < shm_data->length; i++) {
if (memchr(vowels, shm_data->data[i], vowels_count) == NULL) {
output_buffer[output_index++] = shm_data->data[i];
}
}
if (output_index > 0) {
write(output_fd, output_buffer, output_index);
}
shm_data->length = 0;
}
sem_post(sem);
}

sem_close(sem);
munmap(shm_data, SHM_SIZE);
close(shm_fd);
close(output_fd);

return 0;
}
```

Протокол работы программы

```
spr0vay@spr0vayhost:~/Documents/Coding/Labs/OS_Labs/Lab3$ ./server
1.txt
```

```
2.txt
Hello World!!!
It`s Me Marrrrrrioooo
Test1
Test2
Again
Again
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAb
AAAAAAAAAAAAAAAb
AAAAAAAAAAAAAAAA
spr0vay@spr0vayhost:~/Documents/Coding/Labs/OS_Labs/Lab3$ cat 1.txt
Hll Wrld!!!
Tst1
gn

b
spr0vay@spr0vayhost:~/Documents/Coding/Labs/OS_Labs/Lab3$ cat 2.txt
t`s M Mrrrrrr
Tst2
gn
b
```

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 с помощью Shared memory и научился работать с семафорами.