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-216БВ-24

Студент: Лукиянчук А.О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.12.25

Москва, 2025

Постановка задачи

Вариант 16.

Реализовать межпроцессное взаимодействие между родительским (серверным) и дочерним (клиентским) процессами с использованием механизмов разделяемой памяти (shared memory) и семафоров (semaphores). Родительский процесс должен запрашивать у пользователя имя файла, создавать область разделяемой памяти и семафор для синхронизации, затем порождать дочерний процесс. Дочерний процесс должен открывать указанный файл для записи и обрабатывать строки, переданные родительским процессом через разделяемую память. Строка записывается в файл только если она заканчивается на точку (.) или точку с запятой (;), иначе выводится сообщение об ошибке. Взаимодействие должно быть синхронизировано с помощью семафоров.

Общий метод и алгоритм решения

Использованные системные вызовы и функции:

- pid_t fork(void); – создает дочерний процесс.

- int shm_open(const char *name, int oflag, mode_t mode); – создает или открывает объект разделяемой памяти.

- int ftruncate(int fd, off_t length); – устанавливает размер объекта разделяемой памяти.

- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset); – отображает разделяемую память в адресное пространство процесса.

- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создает или открывает именованный семафор.

- int sem_wait(sem_t *sem); – уменьшает значение семафора (операция P).

- int sem_post(sem_t *sem); – увеличивает значение семафора (операция V).

- int sem_close(sem_t *sem); – закрывает семафор.

- int sem_unlink(const char *name); – удаляет именованный семафор из системы.

- int mummap(void *addr, size_t length); – удаляет отображение разделяемой памяти.

- int shm_unlink(const char *name); – удаляет объект разделяемой памяти из системы.

- int excel(const char*path, const char *arg, ...); – заменяет образ текущей программы на указанную.

- pid_t waitpid(pid_t pid, int *status, int options); – ожидает изменения состояния дочернего процесса.

В рамках лабораторной работы создана программа, состоящая из двух процессов: серверного (server) и клиентского (client). Серверный процесс запрашивает у пользователя имя файла, создает область разделяемой памяти и семафор, затем порождает клиентский процесс. Взаимодействие между процессами осуществляется через разделяемую память, синхронизация с помощью семафора. Клиентский процесс проверяет полученные строки на соответствие правилу (окончание на . или ;) и записывает подходящие строки в файл.

Код программы

[src/main.c]

```
#include "../include/server.h"
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
    if (argc < 2) {
        char filename[256];
        const char prompt[] = "Enter filename: ";
        write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

        ssize_t bytes_read = read(STDIN_FILENO, filename, sizeof(filename) - 1);
        if (bytes_read <= 0) {
            const char msg[] = "error: failed to read filename\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }

        if (bytes_read > 0 && filename[bytes_read - 1] == '\n') {
            filename[bytes_read - 1] = '\0';
        } else {
            filename[bytes_read] = '\0';
        }

        run_server_process(filename);
    } else {
        run_server_process(argv[1]);
    }
    return 0;
}
```

[src/client.c]

```
#include "../include/client.h"
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <time.h>

typedef struct {
    char data[4096];
    size_t size;
    int done;
} SharedData;

int check_rule(const char* str) {
    size_t len = strlen(str);
```

```
    if (len == 0) return 0;

    for (size_t i = len - 1; i > 0; --i) {
        if (str[i] != '\n' && str[i] != '\r' && str[i] != ' ' && str[i] != '\t') {
            return (str[i] == '.' || str[i] == ';');
        }
    }

    return (str[0] == '.' || str[0] == ';');
}

void run_client_process(const char* filename, const char* shm_name, const char*
sem_name) {
    int file = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (file == -1) {
        const char error_msg[] = "Error: failed to open file\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);
    }

    int shm_fd = shm_open(shm_name, O_RDWR, 0666);
    if (shm_fd == -1) {
        const char error_msg[] = "Error: failed to open shared memory\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        close(file);
        exit(EXIT_FAILURE);
    }

    SharedData* shared_data = mmap(NULL, sizeof(SharedData),
                                   PROT_READ | PROT_WRITE,
                                   MAP_SHARED, shm_fd, 0);
    if (shared_data == MAP_FAILED) {
        const char error_msg[] = "Error: failed to map shared memory\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        close(shm_fd);
        close(file);
        exit(EXIT_FAILURE);
    }

    close(shm_fd);

    sem_t* semaphore = sem_open(sem_name, 0);
    if (semaphore == SEM_FAILED) {
        const char error_msg[] = "Error: failed to open semaphore\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        munmap(shared_data, sizeof(SharedData));
        close(file);
        exit(EXIT_FAILURE);
    }
```

```
    while (1) {
        sem_wait(semaphore);

        if (shared_data->done == 2) {
            sem_post(semaphore);
            break;
        }

        if (shared_data->size > 0 && shared_data->done == 0) {
            char buffer[4096];
            memcpy(buffer, shared_data->data, shared_data->size);
            buffer[shared_data->size] = '\0';

            char* line = buffer;
            char* next_line;

            while ((next_line = strchr(line, '\n')) != NULL) {
                *next_line = '\0';

                if (check_rule(line)) {
                    size_t line_len = strlen(line);
                    write(file, line, line_len);
                    write(file, "\n", 1);
                    shared_data->done = 1;
                } else {
                    const char error_msg[] = "Error: String does not end with '.' or
';'\n";
                    memcpy(shared_data->data, error_msg, sizeof(error_msg) - 1);
                    shared_data->size = sizeof(error_msg) - 1;
                    shared_data->done = 1;
                }

                line = next_line + 1;
            }

            if (strlen(line) > 0) {
                if (check_rule(line)) {
                    size_t line_len = strlen(line);
                    write(file, line, line_len);
                    write(file, "\n", 1);
                    shared_data->done = 1;
                } else {
                    const char error_msg[] = "Error: String does not end with '.' or
';'\n";
                    memcpy(shared_data->data, error_msg, sizeof(error_msg) - 1);
                    shared_data->size = sizeof(error_msg) - 1;
                    shared_data->done = 1;
                }
            }
```

```
        }

        sem_post(semaphore);

        struct timespec ts;
        ts.tv_sec = 0;
        ts.tv_nsec = 10000000;
        nanosleep(&ts, NULL);
    }

    sem_close(semaphore);
    munmap(shared_data, sizeof(SharedData));
    close(file);
}

int main(int argc, char* argv[]) {
    if (argc < 4) {
        const char error_msg[] = "Error: missing arguments\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        return EXIT_FAILURE;
    }

    run_client_process(argv[1], argv[2], argv[3]);
    return 0;
}
[src/server.c]
#include "../include/server.h"
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/select.h>
#include <semaphore.h>
#include <time.h>

typedef struct {
    char data[4096];
    size_t size;
    int done;
} SharedData;

static void int_to_str(int num, char* str, int max_len) {
    int i = 0;
    int is_negative = 0;

    if (num < 0) {
        is_negative = 1;
```

```
        num = -num;
    }

    do {
        str[i++] = (num % 10) + '0';
        num /= 10;
    } while (num > 0 && i < max_len - 1);

    if (is_negative && i < max_len - 1) {
        str[i++] = '-';
    }

    str[i] = '\0';

    for (int j = 0; j < i / 2; j++) {
        char temp = str[j];
        str[j] = str[i - j - 1];
        str[i - j - 1] = temp;
    }
}

void run_server_process(const char* filename) {
    pid_t pid = getpid();

    char shm_name[256];
    char sem_name[256];

    const char* shm_base = "/lab3_shm_";
    const char* sem_base = "/lab3_sem_";

    int pos = 0;
    const char* p = shm_base;
    while (*p && pos < 255) {
        shm_name[pos++] = *p++;
    }

    char pid_str[32];
    int_to_str(pid, pid_str, sizeof(pid_str));
    p = pid_str;
    while (*p && pos < 255) {
        shm_name[pos++] = *p++;
    }
    shm_name[pos] = '\0';

    pos = 0;
    p = sem_base;
    while (*p && pos < 255) {
        sem_name[pos++] = *p++;
    }
```

```
p = pid_str;
while (*p && pos < 255) {
    sem_name[pos++] = *p++;
}
sem_name[pos] = '\0';

int shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
if (shm_fd == -1) {
    const char msg[] = "error: failed to create shared memory\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (ftruncate(shm_fd, sizeof(SharedData)) == -1) {
    const char msg[] = "error: failed to set shared memory size\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    shm_unlink(shm_name);
    exit(EXIT_FAILURE);
}

SharedData* shared_data = mmap(NULL, sizeof(SharedData),
                               PROT_READ | PROT_WRITE,
                               MAP_SHARED, shm_fd, 0);
if (shared_data == MAP_FAILED) {
    const char msg[] = "error: failed to map shared memory\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    shm_unlink(shm_name);
    exit(EXIT_FAILURE);
}

close(shm_fd);

shared_data->size = 0;
shared_data->done = 0;

sem_t* semaphore = sem_open(sem_name, O_CREAT, 0666, 1);
if (semaphore == SEM_FAILED) {
    const char msg[] = "error: failed to create semaphore\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    munmap(shared_data, sizeof(SharedData));
    shm_unlink(shm_name);
    exit(EXIT_FAILURE);
}

pid_t client_pid = fork();
if (client_pid == -1) {
    const char msg[] = "error: failed to fork\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
```

```
    sem_close(semaphore);
    sem_unlink(sem_name);
    munmap(shared_data, sizeof(SharedData));
    shm_unlink(shm_name);
    exit(EXIT_FAILURE);
}

if (client_pid == 0) {

    char pid_arg[32];
    int_to_str(pid, pid_arg, sizeof(pid_arg));

    execl("./bin/client", "client", filename, shm_name, sem_name, pid_arg, NULL);

    const char msg[] = "error: failed to exec client\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
} else {
    char buffer[4096];
    ssize_t bytes_read;
    int stdin_eof = 0;

    while (!stdin_eof) {
        fd_set readfds;
        FD_ZERO(&readfds);
        FD_SET(STDIN_FILENO, &readfds);

        struct timeval timeout;
        timeout.tv_sec = 0;
        timeout.tv_usec = 100000;

        int activity = select(STDIN_FILENO + 1, &readfds, NULL, NULL, &timeout);

        if (activity < 0) {
            const char msg[] = "error: select failed\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            break;
        }

        if (FD_ISSET(STDIN_FILENO, &readfds)) {
            bytes_read = read(STDIN_FILENO, buffer, sizeof(buffer));
            if (bytes_read > 0) {
                sem_wait(semaphore);
                memcpy(shared_data->data, buffer, bytes_read);
                shared_data->size = bytes_read;
                shared_data->done = 0;
                sem_post(semaphore);

                while (1) {
```

```
                        sem_wait(semaphore);
                        if (shared_data->done == 1) {
                            if (shared_data->size > 0 &&
                                memcmp(shared_data->data, "Error:", 6) == 0) {
                                write(STDERR_FILENO, shared_data->data,
shared_data->size);
                            }
                            sem_post(semaphore);
                            break;
                        }
                        sem_post(semaphore);

                        struct timespec ts;
                        ts.tv_sec = 0;
                        ts.tv_nsec = 10000000;
                        nanosleep(&ts, NULL);
                    }
                } else if (bytes_read == 0) {
                    stdin_eof = 1;
                    sem_wait(semaphore);
                    shared_data->done = 2;
                    sem_post(semaphore);
                }
            }
        }

        int status;
        waitpid(client_pid, &status, 0);

        sem_close(semaphore);
        sem_unlink(sem_name);
        munmap(shared_data, sizeof(SharedData));
        shm_unlink(shm_name);
    }
}
[output.txt]
qwertyu.
qwertyu;
[include/server.h]
#ifndef SERVER_H
#define SERVER_H

void run_server_process(const char* filename);

#endif
[include/client.h]
#ifndef CLIENT_H
#define CLIENT_H
```

```
void run_client_process(const char* filename, const char* shm_name, const char* sem_name);

#endif
```

[Makefile]

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c11 -D_POSIX_C_SOURCE=200809L
LDFLAGS = -lrt -lpthread

SRC_DIR = src
INCLUDE_DIR = include
BIN_DIR = bin
OBJ_DIR = obj

SERVER_SOURCES = $(SRC_DIR)/main.c $(SRC_DIR)/server.c
SERVER_OBJECTS = $(OBJ_DIR)/main.o $(OBJ_DIR)/server.o

CLIENT_SOURCES = $(SRC_DIR)/client.c
CLIENT_OBJECTS = $(OBJ_DIR)/client.o

all: directories $(BIN_DIR)/server $(BIN_DIR)/client

directories:
	mkdir -p $(BIN_DIR) $(OBJ_DIR)

$(BIN_DIR)/server: $(SERVER_OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

$(BIN_DIR)/client: $(CLIENT_OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

$(OBJ_DIR)/main.o: $(SRC_DIR)/main.c $(INCLUDE_DIR)/server.h
	$(CC) $(CFLAGS) -I$(INCLUDE_DIR) -c $< -o $@

$(OBJ_DIR)/server.o: $(SRC_DIR)/server.c $(INCLUDE_DIR)/server.h
	$(CC) $(CFLAGS) -I$(INCLUDE_DIR) -c $< -o $@

$(OBJ_DIR)/client.o: $(SRC_DIR)/client.c $(INCLUDE_DIR)/client.h
	$(CC) $(CFLAGS) -I$(INCLUDE_DIR) -c $< -o $@

clean:
	rm -rf $(BIN_DIR) $(OBJ_DIR)

.PHONY: all clean directories
```

Протокол работы программы

```
[9:27:16] → lab3 ./bin/server output.txt
qwerty.
qwerty;
```

```
      qwerty,
      Error: String does not end with '.' or ';'
      qwerty:
      Error: String does not end with '.' or ';'
      → lab3  cat output.txt
 [9:27:19]
      qwerty.
      qwerty;
→ lab3 strace -f -e trace=clone,futex,fork,execve,mmap,munmap,brk,gettimeofday ./bin/server
output.txt
[14:54:55]
execve("./bin/server", ["./bin/server", "output.txt"], 0x7ffd3f3d3c00 /* 61 vars */) = 0
brk(NULL)                               = 0x55974874e000
mmap(NULL, 166763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb48cb80000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb48cb7e000
mmap(NULL, 2169904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb48c800000
mmap(0x7fb48c824000, 1511424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x24000) = 0x7fb48c824000
mmap(0x7fb48c995000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x195000) =
0x7fb48c995000
mmap(0x7fb48ca04000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x203000) = 0x7fb48ca04000
mmap(0x7fb48ca0a000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
=
0x7fb48ca0a000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb48cb7b000
munmap(0x7fb48cb80000, 166763)          = 0
mmap(NULL, 4112, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fb48cba7000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_DROPPABLE|MAP_ANONYMOUS, -1, 0) = 0x7fb48cba6000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb48cba5000
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fb48cba4000
brk(NULL)                               = 0x55974874e000
brk(0x55974876f000)                     = 0x55974876f000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fb48cb7ba10) = 74404
strace: Process 74404 attached
[pid 74404] execve("./bin/client", ["client", "output.txt", "/lab3_shm_74403",
"/lab3_sem_74403", "74403"], 0x7ffdcac9e500 /*
61 vars */) = 0
[pid 74404] brk(NULL)                   = 0x564dfaf81000
[pid 74404] mmap(NULL, 166763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc83a575000
[pid 74404] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc83a573000
[pid 74404] mmap(NULL, 2169904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc83a200000
[pid 74404] mmap(0x7fc83a224000, 1511424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x24000) = 0x7fc83a224000
[pid 74404] mmap(0x7fc83a395000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x195000) = 0x7fc83a395000
[pid 74404] mmap(0x7fc83a404000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x203000) = 0x7fc83a404000
[pid 74404] mmap(0x7fc83a40a000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
```

```
MAP_ANONYMOUS, -1, 0) = 0x7fc83a40a000
[pid 74404] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc83a570000
[pid 74404] munmap(0x7fc83a575000, 166763) = 0
[pid 74404] mmap(NULL, 4112, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fc83a59c000
[pid 74404] brk(NULL)                   = 0x564dfaf81000
[pid 74404] brk(0x564dfafa2000)         = 0x564dfafa2000
[pid 74404] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fc83a59b000
qwerty;
[pid 74403] futex(0x7fb48cba4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 74404] futex(0x7fc83a59b000, FUTEX_WAKE, 1 <unfinished ...>
[pid 74403] <... futex resumed>)        = 0
[pid 74404] <... futex resumed>)        = 1
qwerty.
qwerty:
Error: String does not end with '.' or ';'
qwerty,
Error: String does not end with '.' or ';'
[pid 74404] munmap(0x7fc83a59b000, 32)  = 0
[pid 74404] munmap(0x7fc83a59c000, 4112) = 0
[pid 74404] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=74404, si_uid=1000, si_status=0,
si_utime=0,
si_stime=2 /* 0.02 s */} ---
munmap(0x7fb48cba4000, 32)              = 0
munmap(0x7fb48cba7000, 4112)            = 0
+++ exited with 0 +++
➜ lab3
```

Вывод

В ходе выполнения лабораторной работы была успешно реализована программа межпроцессного взаимодействия с использованием механизмов разделяемой памяти и семафоров. Отработаны навыки создания и синхронизации процессов, работы с shared memory и именованными семафорами. Программа корректно обрабатывает строки, записывая в файл только те, которые заканчиваются на точку или точку с запятой. Возникшие сложности были связаны с правильной синхронизацией до- ступа к разделяемой памяти и обработкой завершения процессов. Для улучшения программы можно добавить более детальную обработку ошибок и расширить функциональность проверки строк.