**Лабораторная работа номер 5**

Клиент (client.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/types.h>

#include <unistd.h>

#define REQUEST\_MTYPE 1

typedef struct {

long mtype;

pid\_t client\_pid;

char filename[256];

char char\_to\_remove;

} request\_msg;

typedef struct {

long mtype;

int result;

} response\_msg;

int main(int argc, char \*argv[]) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <filename> <char>\n", argv[0]);

exit(EXIT\_FAILURE);

}

if (strlen(argv[2]) != 1) {

fprintf(stderr, "Second argument must be a single character.\n");

exit(EXIT\_FAILURE);

}

key\_t key = ftok("/tmp", 'A');

if (key == -1) {

perror("ftok");

exit(EXIT\_FAILURE);

}

int msgid = msgget(key, 0666 | IPC\_CREAT);

if (msgid == -1) {

perror("msgget");

exit(EXIT\_FAILURE);

}

request\_msg req;

req.mtype = REQUEST\_MTYPE;

req.client\_pid = getpid();

strncpy(req.filename, argv[1], 255);

req.filename[255] = '\0';

req.char\_to\_remove = argv[2][0];

if (msgsnd(msgid, &req, sizeof(req) - sizeof(long), 0) == -1) {

perror("msgsnd");

exit(EXIT\_FAILURE);

}

response\_msg resp;

if (msgrcv(msgid, &resp, sizeof(resp) - sizeof(long), getpid(), 0) == -1) {

perror("msgrcv");

exit(EXIT\_FAILURE);

}

if (resp.result == -1) {

printf("Error processing file.\n");

} else {

printf("Characters removed: %d\n", resp.result);

}

return 0;

}

Сервер (server.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#define REQUEST\_MTYPE 1

typedef struct {

long mtype;

pid\_t client\_pid;

char filename[256];

char char\_to\_remove;

} request\_msg;

typedef struct {

long mtype;

int result;

} response\_msg;

int process\_file(const char \*filename, char char\_to\_remove) {

FILE \*input = fopen(filename, "r");

if (!input) {

return -1;

}

char output\_filename[256];

snprintf(output\_filename, sizeof(output\_filename), "%s.processed", filename);

FILE \*output = fopen(output\_filename, "w");

if (!output) {

fclose(input);

return -1;

}

int count = 0;

int c;

while ((c = fgetc(input)) != EOF) {

if (c != char\_to\_remove) {

fputc(c, output);

} else {

count++;

}

}

fclose(input);

fclose(output);

return count;

}

int main() {

key\_t key = ftok("/tmp", 'A");

if (key == -1) {

perror("ftok");

exit(EXIT\_FAILURE);

}

int msgid = msgget(key, 0666 | IPC\_CREAT);

if (msgid == -1) {

perror("msgget");

exit(EXIT\_FAILURE);

}

printf("Server started. Waiting for requests...\n");

while (1) {

request\_msg req;

if (msgrcv(msgid, &req, sizeof(req) - sizeof(long), REQUEST\_MTYPE, 0) == -1) {

perror("msgrcv");

continue;

}

int result = process\_file(req.filename, req.char\_to\_remove);

response\_msg resp;

resp.mtype = req.client\_pid;

resp.result = result;

if (msgsnd(msgid, &resp, sizeof(resp) - sizeof(long), 0) == -1) {

perror("msgsnd");

}

}

return 0;

}

Описание работы

* **Клиент** принимает два аргумента: имя файла и символ для удаления. Он отправляет запрос серверу через очередь сообщений.
* **Сервер** ожидает запросы, обрабатывает файл, удаляя указанный символ, и сохраняет результат в новый файл с расширением .processed. Результат (количество удаленных символов или -1 при ошибке) отправляется обратно клиенту.
* **Обработка ошибок** включает проверку аргументов, доступность файлов и корректность работы с очередями сообщений.

**Лабораторная работа номер 6**

Для реализации задачи удаления заданного символа из текста с использованием **разделяемой памяти и семафоров System V**, мы создадим две программы: **клиент (родительский процесс)** и **сервер (дочерний процесс)**. Клиент передает данные через разделяемую память, а сервер обрабатывает их, удаляя указанный символ.

## **1. Клиент (client.c)**

Отправляет данные серверу через разделяемую память и синхронизирует работу с помощью семафоров.

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <sys/sem.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#define SHM\_SIZE 1024**

**// Структура для данных в разделяемой памяти**

**typedef struct {**

**char filename[256];**

**char char\_to\_remove;**

**char processed\_data[SHM\_SIZE];**

**int result;**

**} shared\_data;**

**// Операции с семафором**

**void sem\_wait(int semid) {**

**struct sembuf op = {0, -1, 0};**

**semop(semid, &op, 1);**

**}**

**void sem\_signal(int semid) {**

**struct sembuf op = {0, 1, 0};**

**semop(semid, &op, 1);**

**}**

**int main(int argc, char \*argv[]) {**

**if (argc != 3) {**

**fprintf(stderr, "Usage: %s <filename> <char>\n", argv[0]);**

**exit(EXIT\_FAILURE);**

**}**

**if (strlen(argv[2]) != 1) {**

**fprintf(stderr, "Second argument must be a single character.\n");**

**exit(EXIT\_FAILURE);**

**}**

**// Создаем ключ для разделяемой памяти и семафора**

**key\_t key = ftok("/tmp", 'A');**

**if (key == -1) {**

**perror("ftok");**

**exit(EXIT\_FAILURE);**

**}**

**// Создаем разделяемую память**

**int shmid = shmget(key, sizeof(shared\_data), IPC\_CREAT | 0666);**

**if (shmid == -1) {**

**perror("shmget");**

**exit(EXIT\_FAILURE);**

**}**

**// Подключаем разделяемую память**

**shared\_data \*shm\_data = (shared\_data \*)shmat(shmid, NULL, 0);**

**if (shm\_data == (void \*)-1) {**

**perror("shmat");**

**exit(EXIT\_FAILURE);**

**}**

**// Создаем семафор**

**int semid = semget(key, 1, IPC\_CREAT | 0666);**

**if (semid == -1) {**

**perror("semget");**

**exit(EXIT\_FAILURE);**

**}**

**// Инициализируем семафор значением 0 (ожидание сервера)**

**semctl(semid, 0, SETVAL, 0);**

**// Заполняем данные в разделяемой памяти**

**strncpy(shm\_data->filename, argv[1], 255);**

**shm\_data->char\_to\_remove = argv[2][0];**

**shm\_data->result = -1;**

**// Запускаем сервер (дочерний процесс)**

**pid\_t pid = fork();**

**if (pid == -1) {**

**perror("fork");**

**exit(EXIT\_FAILURE);**

**} else if (pid == 0) {**

**// Дочерний процесс (сервер)**

**execl("./server", "server", NULL);**

**perror("execl");**

**exit(EXIT\_FAILURE);**

**}**

**// Ждем, пока сервер обработает данные**

**sem\_wait(semid);**

**// Выводим результат**

**if (shm\_data->result == -1) {**

**printf("Error processing file.\n");**

**} else {**

**printf("Characters removed: %d\n", shm\_data->result);**

**printf("Processed data:\n%s\n", shm\_data->processed\_data);**

**}**

**// Освобождаем ресурсы**

**shmdt(shm\_data);**

**shmctl(shmid, IPC\_RMID, NULL);**

**semctl(semid, 0, IPC\_RMID);**

**return 0;**

**}**

## **2. Сервер (server.c)**

Получает данные из разделяемой памяти, обрабатывает их и возвращает результат.

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <sys/sem.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#define SHM\_SIZE 1024**

**typedef struct {**

**char filename[256];**

**char char\_to\_remove;**

**char processed\_data[SHM\_SIZE];**

**int result;**

**} shared\_data;**

**void sem\_wait(int semid) {**

**struct sembuf op = {0, -1, 0};**

**semop(semid, &op, 1);**

**}**

**void sem\_signal(int semid) {**

**struct sembuf op = {0, 1, 0};**

**semop(semid, &op, 1);**

**}**

**int main() {**

**// Получаем ключ**

**key\_t key = ftok("/tmp", 'A');**

**if (key == -1) {**

**perror("ftok");**

**exit(EXIT\_FAILURE);**

**}**

**// Подключаемся к разделяемой памяти**

**int shmid = shmget(key, sizeof(shared\_data), 0666);**

**if (shmid == -1) {**

**perror("shmget");**

**exit(EXIT\_FAILURE);**

**}**

**shared\_data \*shm\_data = (shared\_data \*)shmat(shmid, NULL, 0);**

**if (shm\_data == (void \*)-1) {**

**perror("shmat");**

**exit(EXIT\_FAILURE);**

**}**

**// Получаем семафор**

**int semid = semget(key, 1, 0666);**

**if (semid == -1) {**

**perror("semget");**

**exit(EXIT\_FAILURE);**

**}**

**// Обрабатываем файл**

**FILE \*file = fopen(shm\_data->filename, "r");**

**if (!file) {**

**shm\_data->result = -1;**

**sem\_signal(semid); // Разблокируем клиента**

**exit(EXIT\_FAILURE);**

**}**

**int count = 0;**

**char c;**

**int i = 0;**

**while ((c = fgetc(file)) != EOF) {**

**if (c != shm\_data->char\_to\_remove) {**

**shm\_data->processed\_data[i++] = c;**

**} else {**

**count++;**

**}**

**}**

**shm\_data->processed\_data[i] = '\0';**

**shm\_data->result = count;**

**fclose(file);**

**// Сигнализируем клиенту, что данные готовы**

**sem\_signal(semid);**

**// Отключаемся от разделяемой памяти**

**shmdt(shm\_data);**

**return 0;**

**}**

## **Объяснение работы**

1. **Клиент:**
   * Создает **разделяемую память** и **семафор**.
   * Записывает в разделяемую память имя файла и символ для удаления.
   * Запускает **сервер** (дочерний процесс).
   * Ждет, пока сервер обработает данные (блокировка через семафор).
2. **Сервер:**
   * Читает данные из разделяемой памяти.
   * Удаляет указанный символ из файла.
   * Сохраняет результат в разделяемую память.
   * Разблокирует клиент через семафор.
3. **Синхронизация:**
   * Используется **двоичный семафор** (0 — заблокирован, 1 — разблокирован).
   * Клиент ждет (sem\_wait), пока сервер не обработает данные и не подаст сигнал (sem\_signal).

## **Обработка ошибок**

* Проверка аргументов командной строки.
* Обработка ошибок при работе с файлами.
* Ошибки при создании/подключении разделяемой памяти и семафоров.

Этот код демонстрирует **межпроцессное взаимодействие через разделяемую память с синхронизацией на семафорах System V**.

**Лабораторная работа №7 «Символьные устройства».**

## Общий план реализации

1. **Функция обработки (потока)**  
   Функция, которая для одного файла и символа: читает файл, удаляет все вхождения заданного символа, записывает результат в отдельный выходной файл и возвращает количество удалённых символов (или -1 при ошибке).
2. **Главная функция (main)**
   * Принимает аргументы командной строки: список файлов и символ для удаления.
   * Создаёт по одному потоку на каждый входной файл (например, с помощью pthread\_create).
   * Ждёт завершения всех потоков и выводит их коды завершения.
   * Обрабатывает ошибки.
3. **Синхронизация**  
   Для обмена информацией — например, глобальный массив структур, защищённый мьютексами.
4. **Обработка исключений**
   * Проверка аргументов командной строки.
   * Проверка успешности создания потока, открытия файлов и работы с мьютексами.

#include <stdio.h> #include <stdlib.h> #include <pthread.h> #include <string.h> #include <errno.h>

typedef struct { const char \*input\_file; char symbol\_to\_remove; int operations\_count; // количество удалённых символов или -1 при ошибке } thread\_data\_t;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*process\_file(void \*arg) { thread\_data\_t \*data = (thread\_data\_t \*)arg; FILE \*fin = fopen(data->input\_file, "r"); if (!fin) { perror("Ошибка открытия входного файла"); data->operations\_count = -1; pthread\_exit(NULL); }

// Формируем имя выходного файла (например, input.txt -> input\_out.txt)

char out\_filename[256];

snprintf(out\_filename, sizeof(out\_filename), "%s\_out.txt", data->input\_file);

FILE \*fout = fopen(out\_filename, "w");

if (!fout) {

perror("Ошибка открытия выходного файла");

fclose(fin);

data->operations\_count = -1;

pthread\_exit(NULL);

}

int ch;

int removed = 0;

while ((ch = fgetc(fin)) != EOF) {

if (ch == data->symbol\_to\_remove) {

removed++;

continue;

}

fputc(ch, fout);

}

fclose(fin);

fclose(fout);

data->operations\_count = removed;

pthread\_exit(NULL);

}

int main(int argc, char \*argv[]) { if (argc < 3) { fprintf(stderr, "Использование: %s символ\_удаления файл1 [файл2 ...]\n", argv[0]); return EXIT\_FAILURE; }

char symbol = argv[1][0]; // символ для удаления

int file\_count = argc - 2;

pthread\_t \*threads = malloc(file\_count \* sizeof(pthread\_t));

thread\_data\_t \*thread\_data = malloc(file\_count \* sizeof(thread\_data\_t));

if (!threads || !thread\_data) {

fprintf(stderr, "Ошибка выделения памяти\n");

free(threads);

free(thread\_data);

return EXIT\_FAILURE;

}

for (int i = 0; i < file\_count; i++) {

thread\_data[i].input\_file = argv[i + 2];

thread\_data[i].symbol\_to\_remove = symbol;

thread\_data[i].operations\_count = 0;

int res = pthread\_create(&threads[i], NULL, process\_file, &thread\_data[i]);

if (res != 0) {

fprintf(stderr, "Ошибка создания потока для файла %s: %s\n",

thread\_data[i].input\_file, strerror(res));

thread\_data[i].operations\_count = -1;

}

}

for (int i = 0; i < file\_count; i++) {

void \*retval;

int res = pthread\_join(threads[i], &retval);

if (res != 0) {

fprintf(stderr, "Ошибка ожидания потока %d: %s\n", i, strerror(res));

} else {

printf("Поток для файла %s завершился с результатом: %d\n",

thread\_data[i].input\_file, thread\_data[i].operations\_count);

}

}

free(threads);

free(thread\_data);

return EXIT\_SUCCESS;

}

**Лабораторная работа No 8  
Сетевое взаимодействие процессов в Linux**

* **Вариант** — удалить из текста заданный символ (как и в вашей предыдущей работе с файлами).
* **Сетевой протокол:**
  + Чётный вариант — UDP
  + Нечётный вариант — TCP  
    (Выберите в зависимости от вашего варианта; если не указано — берите TCP.)
* Сервер должен:
  + Получить от клиента имя файла и символ для удаления.
  + Открыть файл (локально у себя), сделать обработку (удалить символ).
  + Записать результат в новый файл и вернуть клиенту количество удалений (или -1 при ошибке).
  + Вести логирование через syslog.
  + Работать как демон (сначала без демона, потом добавить преобразование в демон).
* Клиент должен:
  + Послать серверу имя файла и символ.
  + Получить результат.
  + При необходимости, сохранить/вывести результат.

**Ключевые компоненты**

* **Сервер:**
  + Создаёт TCP или UDP сокет и слушает входящие запросы.
  + Для каждого запроса запускает обработку файла.
  + Ведёт лог через syslog() вместо printf.
  + Позже запускается как демон — с fork(), перекрытием stdio, вызовом setsid(), и пр.
* **Клиент:**
  + Подключается к серверу через TCP или отправляет UDP-пакеты.
  + Отправляет данные (имя файла, символ).
  + Получает ответ (число удалённых символов или -1).
  + Выводит результат или сохраняет его.

**Серверная часть TCP**

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <syslog.h> #include <sys/socket.h> #include <netinet/in.h>

#define PORT 12345 #define BUF\_SIZE 1024

int process\_file(const char \*filename, char symbol, char \*output\_file) { FILE \*fin = fopen(filename, "r"); if (!fin) { syslog(LOG\_ERR, "Ошибка открытия файла %s", filename); return -1; }

snprintf(output\_file, 256, "%s\_out.txt", filename);

FILE \*fout = fopen(output\_file, "w");

if (!fout) {

syslog(LOG\_ERR, "Ошибка создания выходного файла");

fclose(fin);

return -1;

}

int ch, removed = 0;

while ((ch = fgetc(fin)) != EOF) {

if (ch == symbol) {

removed++;

} else {

fputc(ch, fout);

}

}

fclose(fin);

fclose(fout);

syslog(LOG\_INFO, "Файл %s обработан, удалено %d символов '%c'", filename, removed, symbol);

return removed;

}

int main() { openlog("myserver", LOG\_PID | LOG\_CONS, LOG\_USER);

int server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd < 0) {

syslog(LOG\_ERR, "Ошибка создания сокета");

return 1;

}

struct sockaddr\_in addr = {0};

addr.sin\_family = AF\_INET;

addr.sin\_addr.s\_addr = INADDR\_ANY;

addr.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr\*)&addr, sizeof(addr)) < 0) {

syslog(LOG\_ERR, "Ошибка привязки сокета");

close(server\_fd);

return 1;

}

if (listen(server\_fd, 5) < 0) {

syslog(LOG\_ERR, "Ошибка listen");

close(server\_fd);

return 1;

}

syslog(LOG\_INFO, "Сервер запущен, слушает порт %d", PORT);

while (1) {

struct sockaddr\_in client\_addr;

socklen\_t client\_len = sizeof(client\_addr);

int client\_fd = accept(server\_fd, (struct sockaddr\*)&client\_addr, &client\_len);

if (client\_fd < 0) {

syslog(LOG\_ERR, "Ошибка accept");

continue;

}

// Получаем данные от клиента

char buffer[BUF\_SIZE];

ssize\_t n = read(client\_fd, buffer, sizeof(buffer) - 1);

if (n <= 0) {

syslog(LOG\_ERR, "Ошибка чтения данных от клиента");

close(client\_fd);

continue;

}

buffer[n] = '\0';

// Формат данных: "filename;symbol"

char \*filename = strtok(buffer, ";");

char \*symbol\_str = strtok(NULL, ";");

if (!filename || !symbol\_str || strlen(symbol\_str) != 1) {

syslog(LOG\_ERR, "Некорректные данные от клиента");

close(client\_fd);

continue;

}

char output\_file[256];

int result = process\_file(filename, symbol\_str[0], output\_file);

// Отправляем результат клиенту

char reply[256];

snprintf(reply, sizeof(reply), "%d", result);

write(client\_fd, reply, strlen(reply));

close(client\_fd);

}

closelog();

close(server\_fd);

return 0;

}

**Клиентская часть TCP**

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <arpa/inet.h>

#define PORT 12345 #define BUF\_SIZE 1024

int main(int argc, char \*argv[]) { if (argc != 3) { fprintf(stderr, "Использование: %s имя\_файла символ\n", argv[0]); return 1; }

int sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Ошибка создания сокета");

return 1;

}

struct sockaddr\_in server\_addr = {0};

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr);

if (connect(sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Ошибка подключения к серверу");

close(sock);

return 1;

}

char msg[BUF\_SIZE];

snprintf(msg, sizeof(msg), "%s;%s", argv[1], argv[2]);

write(sock, msg, strlen(msg));

char buffer[BUF\_SIZE];

ssize\_t n = read(sock, buffer, sizeof(buffer) - 1);

if (n <= 0) {

perror("Ошибка чтения ответа");

close(sock);

return 1;

}

buffer[n] = '\0';

printf("Количество удалённых символов: %s\n", buffer);

close(sock);

return 0;

}