Федеральное государственное бюджетное образовательное учреждение высшего образования

Национально исследовательский университет «Московский энергетический институт»

Кафедра ВМСС

Лабораторная работа №2

по курсу

«Программное обеспечение высокопроизводительных вычислительных систем»

Выполнили: студенты группы А-07м-23

Балашов С.А., Кретов Н.В., Михайлов С.А.

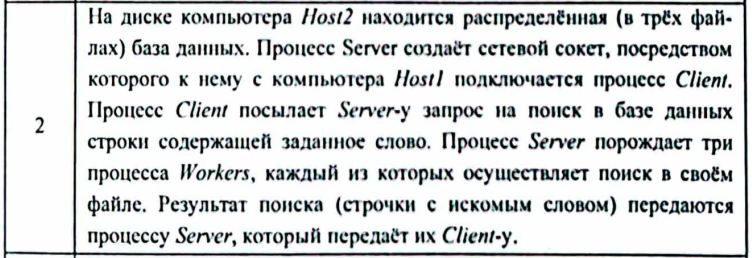
Проверил: Орлов Д.А., Хиль С.

Москва, 2024 г.

**Задание:**

Разработать программы для комплекса процессов согласно варианту задания.

**Вариант 2:**

****

**Файл client.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define msgSize 256

/// Структура для настройки сокета

struct ip\_config

{

char address[16];

uint16\_t port;

};

int main()

{

/// Настройка переменных

struct ip\_config config = { "127.0.0.1", 3245 };// Адрес и порт сервера

char MsgToServer[msgSize], MsgFromServer[msgSize]; // Сообщения для обмена

int sockid; // Идентификатор сокета

struct sockaddr\_in addr; // Структура сетевого адреса сокета

sockid = socket(AF\_INET, SOCK\_STREAM, 0); // Создание сокета

addr.sin\_family = AF\_INET; // Присвоение типа

addr.sin\_port = htons(config.port);// Номер сетевого порта

inet\_aton(config.address, &addr.sin\_addr); // Адрес

/// Попытка соединения

if (connect(sockid, &addr, sizeof(addr)) < 0)

{

printf("Connection error\n");

return 1;

}

/// Отправка сообщения серверу

printf("Введите слово для поиска:");

scanf("%s", MsgToServer);

write(sockid, MsgToServer, msgSize);

/// Приём сообщения от сервера

printf("Номер процесса и строка с заданным словом: ");

read(sockid, MsgFromServer, msgSize);

printf("%s\n", MsgFromServer);

close(sockid);

return 0;

}

**Файл server.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/un.h>

#define msgSize 256

/// Структура для настройки сокета

struct ip\_config

{

char address[16];

uint16\_t port;

};

/// Процедура предка для распределения данных между потомками и отправки результата клиенту

void controller(int pipe\_fd[6][2], char \*mem, int socket\_id);

/// Процедура потомка для обработки данных

void worker(int pipefd\_in\_1, int pipefd\_out\_1, int pipefd\_in\_2, int pipefd\_out\_2, char \*mem, FILE \*file);

int main()

{

/// Настройка переменных

struct ip\_config config = { "127.0.0.1", 3245 };// Адрес и порт сервера

char path[3][15] = { "../file1.txt", "../file2.txt", "../file3.txt" }; // Пути к файлам базы данных

char mem[msgSize]; // Буфер для обмена данными

int socket\_fd, new\_socket\_fd, pipe\_fd[6][2]; // Файлы сокета и каналов

FILE \*file1, \*file2, \*file3; // Файлы базы данных

pid\_t w1\_pid, w2\_pid, w3\_pid; // Идентификаторы потомков

/// Создание сокета

struct sockaddr\_in address;

address.sin\_family = AF\_INET; // Присвоение типа семейства сокетов

address.sin\_port = htons(config.port); // Номер сетевого порта

inet\_aton(config.address, &address.sin\_addr); // Адрес

socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // Создание сокета

if (socket\_fd < 0)

{

perror("socket");

exit(1);

}

/// Привязка сокета

int bind\_status = bind(

socket\_fd,

(const struct sockaddr \*) &address, sizeof(address)

);

if (bind\_status < 0)

{

perror("bind");

exit(1);

}

/// Прослушивание сокета и принятие соединения

listen(socket\_fd, 1);

new\_socket\_fd = accept(socket\_fd, NULL, NULL);

/// Создание каналов

for (int i = 0; i < 6; ++i) {

if(pipe(pipe\_fd[i]) < 0) {

perror("pipe");

exit(1);

}

}

/// Открытие файлов

file3 = fopen(path[2], "r");

if (file3 == NULL)

{

perror("file3");

exit(1);

}

file2 = fopen(path[1], "r");

if (file2 == NULL)

{

perror("file2");

exit(1);

}

file1 = fopen(path[0], "r");

if (file1 == NULL)

{

perror("file1");

exit(1);

}

/// Запуск потомков и процедуры предка

if ((w1\_pid = fork()) != 0)

{

if ((w2\_pid = fork()) != 0)

{

if ((w3\_pid = fork()) != 0)

{

/// Процедура предка

controller(pipe\_fd, mem, new\_socket\_fd);

/// Закрытие сокета, каналов и потомков

close(socket\_fd);

for (int i = 0; i < 6; ++i) {

close(pipe\_fd[i][0]);

close(pipe\_fd[i][1]);

}

waitpid(w1\_pid, NULL, 0);

waitpid(w2\_pid, NULL, 0);

waitpid(w3\_pid, NULL, 0);

}else

{

/// Потомок 3

worker(pipe\_fd[4][0], pipe\_fd[4][1], pipe\_fd[5][0], pipe\_fd[5][1], mem, file3);

}

} else

{

/// Потомок 2

worker(pipe\_fd[2][0], pipe\_fd[2][1], pipe\_fd[3][0], pipe\_fd[3][1], mem, file2);

}

} else

{

/// Потомок 1

worker(pipe\_fd[0][0], pipe\_fd[0][1], pipe\_fd[1][0], pipe\_fd[1][1], mem, file1);

}

return 0;

}

void worker(int pipefd\_in\_1, int pipefd\_out\_1, int pipefd\_in\_2, int pipefd\_out\_2, char \*mem, FILE \*file) {

char str[msgSize], \*str2 = NULL;

/// Чтение искомого слова из канала

close(pipefd\_in\_2);

close(pipefd\_out\_1);

read(pipefd\_in\_1, mem, msgSize);

close(pipefd\_in\_1);

/// Поиск искомого слова

while (fgets(str, msgSize, file) != NULL)

{

str2 = strstr(str, mem);

if (str2 != NULL)

{

/// Запись результата в канал

char s[msgSize];

sprintf(s,"%d - %s", getpid(), str);

write(pipefd\_out\_2, s, msgSize);

close(pipefd\_out\_2);

fclose(file);

break;

}

}

fclose(file);

close(pipefd\_out\_2);

exit(0);

}

void controller(int pipe\_fd[6][2], char \*mem, int socket\_id) {

/// Получение искомого слова от клиента

read(socket\_id, mem, msgSize);

close(pipe\_fd[0][0]);

close(pipe\_fd[2][0]);

close(pipe\_fd[4][0]);

/// Запись искомого слова в канал потомка 1

write(pipe\_fd[0][1], mem, msgSize);

close(pipe\_fd[0][1]);

/// Запись искомого слова в канал потомка 2

write(pipe\_fd[2][1], mem, msgSize);

close(pipe\_fd[2][1]);

/// Запись искомого слова в канал потомка 1

write(pipe\_fd[4][1], mem, msgSize);

close(pipe\_fd[4][1]);

char found[3][msgSize];

close(pipe\_fd[1][1]);

close(pipe\_fd[3][1]);

close(pipe\_fd[5][1]);

/// Чтение результата из канала потомка 1

read(pipe\_fd[1][0], found[0], msgSize);

close(pipe\_fd[1][0]);

/// Чтение результата из канала потомка 2

read(pipe\_fd[3][0], found[0], msgSize);

close(pipe\_fd[3][0]);

/// Чтение результата из канала потомка 3

read(pipe\_fd[5][0], found[0], msgSize);

close(pipe\_fd[5][0]);

char msg\_to\_client[msgSize];

/// Формирование результата

for (int i = 0; i < 2; ++i) {

if(found[i] != NULL)

{

sprintf(msg\_to\_client, "%s", found[i]);

break;

}

}

/// Отправка результата и закрытие сокета

write(socket\_id, msg\_to\_client, msgSize);

close(socket\_id);

}