**Операционные системы**

# **Отчёт по ИДЗ №3**

Максимов Никита Иванович

БПИ227

19.05.2024

Работа на оценку 8

## Вариант 7

## Условие:

**Задача о читателях и писателях («подтвержденное чтение»).** Базу данных, представленную массивом целых положительных чисел, разделяют два типа процессов: N читателей и K писателей. Читатели периодически просматривают случайные записи базы данных и выводя номер свой номер (например, PID), индекс записи, ее значение, а также вычисленное значение, которое является произведением числа на номер записи. Писатели изменяют случайные записи на случайное число и также выводят информацию о своем номере, индексе записи, старом значении и новом значении. Предполагается, что в начале БД находится в непротиворечивом состоянии (все числа отсортированы). Каждая отдельная новая запись переводит БД из одного непротиворечивого состояния в другое (то есть, новая сортировка может поменять индексы записей или переставить числа). Транзакции выполняются в режиме «подтвержденного чтения», то есть процесс-писатель не может получить доступ к БД в том случае, если ее занял другой процесс–писатель или процесс–читатель. К БД может обратиться одновременно сколько угодно процессов–читателей. Процесс читатель получает доступ к БД, даже если ее уже занял процесс–писатель.

Создать клиент–серверное приложение с процессами–писателями и процессами–читателями. Сервер моделирует базу данных. Все писатели и все читатели — два разных клиента, в каждом из которых возможна конкуренция параллельных процессов или потоков. Каждый процесс — это отдельный писатель или отдельный читатель внутри сервера.

## Структура

Программа состоит из одного сервера, клиента читателей, клиента писателей, и выводящего клиента. Каждый из них запускается по отдельности.

Сервер представляет из себя базу данных, где находится информация о книгах. Сервер постоянно слушает читателей и писателей на получение новых сообщений. Все действия он фиксирует в лог и передаёт выводящей программе.

Клиент читателей – 5 сущностей, постоянно берущие случайный индекс из библиотеки и передающие свой номер и номер индекса на сервер, где сервер уже считает нужное значение (произведение книги на её порядковый номер).

Клиент писателей – 5 сущностей, постоянно берущие случайный индекс из библиотеки и передающие свой номер и номер индекса на сервер, где сервер записывает новое значение на индекс и сортирует библиотеку алгоритмом **Selection Sort**.

Клиент выводящей программы постоянно читает сокет и выводит то, что прочитал в очередной раз.**Запуск**

## Программа запускается через 4 разных терминала следующими командами:

## *gcc server.c -o server.out && ./server.out*

## *gcc readers.c -o readers.out && ./readers.out*

## *gcc writers.c -o writers.out && ./writers.out*

## *gcc log.c -o log.out && ./log.out*

## 

## Текст программы

**Файл “server.c”**

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

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <signal.h>**

**#include <string.h>**

**#define LIBRARY\_SIZE 20**

**#define PORT 8888**

**#define MAX\_CLIENTS 3**

**void swap(int\* xp, int\* yp) {**

**int temp = \*xp;**

**\*xp = \*yp;**

**\*yp = temp;**

**}**

**void selectionSort(int arr[]) {**

**int min\_idx;**

**for (int i = 0; i < LIBRARY\_SIZE - 1; ++i) {**

**min\_idx = i;**

**for (int j = i + 1; j < LIBRARY\_SIZE; ++j) {**

**if (arr[j] < arr[min\_idx]) {**

**min\_idx = j;**

**}**

**}**

**swap(&arr[min\_idx], &arr[i]);**

**}**

**}**

**void join(int\* arr, char\* buffer) {**

**int offset = 0;**

**for (int i = 0; i < LIBRARY\_SIZE; ++i) {**

**int length = snprintf(NULL, 0, "%d", arr[i]);**

**char temp[length + 1];**

**snprintf(temp, length + 1, "%d", arr[i]);**

**if (i > 0) {**

**buffer[offset] = ' ';**

**++offset;**

**}**

**for (int j = 0; j < length; ++j) {**

**buffer[offset] = temp[j];**

**++offset;**

**}**

**}**

**buffer[offset] = '\n';**

**buffer[offset + 1] = '\0';**

**}**

**int library[LIBRARY\_SIZE];**

**sig\_atomic\_t killed = 0;**

**void killing\_handler(int sig) {**

**killed = 1;**

**}**

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

**int client\_socket[MAX\_CLIENTS];**

**for (int i = 0; i < MAX\_CLIENTS; ++i) {**

**client\_socket[i] = 0;**

**}**

**int master\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (master\_socket == -1) {**

**perror("socket failed");**

**exit(EXIT\_FAILURE);**

**}**

**int opt = 1;**

**if (setsockopt(master\_socket, SOL\_SOCKET, SO\_REUSEADDR, (char \*) &opt, sizeof(opt)) == -1) {**

**perror("setsockopt failed");**

**exit(EXIT\_FAILURE);**

**}**

**struct sockaddr\_in addr = { 0 };**

**if (argc == 2) {**

**addr.sin\_addr.s\_addr = inet\_addr(argv[0]);**

**}**

**addr.sin\_family = AF\_INET;**

**addr.sin\_port = htons(PORT);**

**if (argc == 2) {**

**int port;**

**sscanf(argv[1], "%d", &port);**

**addr.sin\_port = htons(port);**

**}**

**int bind\_res = bind(master\_socket, (struct sockaddr \*) & addr, sizeof addr);**

**if (bind\_res == -1) {**

**perror("bind failed");**

**exit(EXIT\_FAILURE);**

**}**

**int listen\_res = listen(master\_socket, MAX\_CLIENTS);**

**if (listen\_res == -1) {**

**perror("listen failed");**

**exit(EXIT\_FAILURE);**

**}**

**socklen\_t addrlen = sizeof addr;**

**for (int i = 0; i < MAX\_CLIENTS; ++i) {**

**client\_socket[i] = accept(master\_socket, (struct sockaddr\*)&addr, &addrlen);**

**if (client\_socket[i] == -1) {**

**perror("accept failed");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**ssize\_t nread;**

**char buf[256];**

**signal(SIGINT, killing\_handler);**

**for (int i = 0; i < LIBRARY\_SIZE; ++i) {**

**library[i] = i + 1;**

**}**

**sprintf(buf, "Library is initialized\n");**

**write(client\_socket[2], buf, 256);**

**// i == 0: readers**

**// i == 1: writers**

**for (int i = 0; i < MAX\_CLIENTS - 1; ++i) {**

**if (fork() != 0) {**

**continue;**

**}**

**if (i == 0) {**

**while (1) {**

**if (killed) {**

**for (int k = 0; k < MAX\_CLIENTS; ++k) {**

**close(client\_socket[k]);**

**}**

**close(listen\_res);**

**close(master\_socket);**

**printf("Все клиенты и сервер закрыты\n");**

**exit(0);**

**}**

**nread = read(client\_socket[i], buf, 256);**

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

**perror("read failed");**

**exit(EXIT\_FAILURE);**

**}**

**if (nread == 0) {**

**printf("END OF FILE occured\n");**

**continue;**

**}**

**int reader;**

**sscanf(buf, "%d", &reader);**

**nread = read(client\_socket[i], buf, 256);**

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

**perror("read failed");**

**exit(EXIT\_FAILURE);**

**}**

**if (nread == 0) {**

**printf("END OF FILE occured\n");**

**continue;**

**}**

**int book;**

**sscanf(buf, "%d", &book);**

**sprintf(buf, "Reader #%d read the book #%d. Result: %d \* %d = %d\n",**

**reader + 1, book + 1, book + 1, library[book], (book + 1) \* (library[book]));**

**write(client\_socket[2], buf, sizeof(buf));**

**join(library, buf);**

**write(client\_socket[2], buf, sizeof(buf));**

**}**

**}**

**else if (i == 1) {**

**while (1) {**

**if (killed) {**

**for (int k = 0; k < MAX\_CLIENTS; ++k) {**

**close(client\_socket[k]);**

**}**

**close(listen\_res);**

**close(master\_socket);**

**printf("Все клиенты и сервер закрыты\n");**

**exit(0);**

**}**

**nread = read(client\_socket[i], buf, 256);**

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

**perror("read failed");**

**exit(EXIT\_FAILURE);**

**}**

**if (nread == 0) {**

**printf("END OF FILE occured\n");**

**continue;**

**}**

**int writer;**

**sscanf(buf, "%d", &writer);**

**nread = read(client\_socket[i], buf, 256);**

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

**perror("read failed");**

**exit(EXIT\_FAILURE);**

**}**

**if (nread == 0) {**

**printf("END OF FILE occured\n");**

**continue;**

**}**

**int book;**

**sscanf(buf, "%d", &book);**

**int new\_book = rand() % 30;**

**sprintf(buf, "Writer #%d written the book #%d. It was %d: now it's %d\n",**

**writer + 1, book + 1, library[book], new\_book);**

**write(client\_socket[2], buf, sizeof(buf));**

**library[book] = new\_book;**

**selectionSort(library);**

**join(library, buf);**

**write(client\_socket[2], buf, sizeof(buf));**

**}**

**}**

**exit(0);**

**}**

**return 0;**

**}**

## 

## Файл “readers.c”

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

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <signal.h>**

**#include <string.h>**

**#define LIBRARY\_SIZE 20**

**#define PORT 8888**

**#define READERS\_SIZE 5**

**sig\_atomic\_t killed = 0;**

**void killing\_handler(int sig) {**

**killed = 1;**

**}**

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

**int fd\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (fd\_socket == -1) {**

**perror("socket failed");**

**exit(EXIT\_FAILURE);**

**}**

**struct sockaddr\_in addr = { 0 };**

**if (argc == 2) {**

**addr.sin\_addr.s\_addr = inet\_addr(argv[0]);**

**}**

**addr.sin\_family = AF\_INET;**

**addr.sin\_port = htons(PORT);**

**if (argc == 2) {**

**int port;**

**sscanf(argv[1], "%d", &port);**

**addr.sin\_port = htons(port);**

**}**

**int connect\_res = connect(fd\_socket, (struct sockaddr \*) &addr, sizeof addr);**

**if (connect\_res == -1) {**

**perror("connect failed");**

**exit(EXIT\_FAILURE);**

**}**

**ssize\_t nread;**

**signal(SIGINT, killing\_handler);**

**char buf[256];**

**for (int i = 0; i < READERS\_SIZE; ++i) {**

**if (fork() == 0) {**

**while (1) {**

**if (killed) {**

**printf("Клиент читателя закрыт\n");**

**close(fd\_socket);**

**exit(0);**

**}**

**// отправляет номер читателя**

**sprintf(buf, "%d", i);**

**write(fd\_socket, buf, 256);**

**// отправляет номер книги**

**sprintf(buf, "%d", rand() % LIBRARY\_SIZE);**

**write(fd\_socket, buf, 256);**

**sleep(rand() % 2 + 2);**

**}**

**exit(0);**

**}**

**}**

**return 0;**

**}**

## Файл “writers.c”

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

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <signal.h>**

**#include <string.h>**

**#define LIBRARY\_SIZE 20**

**#define PORT 8888**

**#define WRITERS\_SIZE 5**

**sig\_atomic\_t killed = 0;**

**void killing\_handler(int sig) {**

**killed = 1;**

**}**

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

**int fd\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (fd\_socket == -1) {**

**perror("socket failed");**

**exit(EXIT\_FAILURE);**

**}**

**struct sockaddr\_in addr = { 0 };**

**if (argc == 2) {**

**addr.sin\_addr.s\_addr = inet\_addr(argv[0]);**

**}**

**addr.sin\_family = AF\_INET;**

**addr.sin\_port = htons(PORT);**

**if (argc == 2) {**

**int port;**

**sscanf(argv[1], "%d", &port);**

**addr.sin\_port = htons(port);**

**}**

**int connect\_res = connect(fd\_socket, (struct sockaddr \*) &addr, sizeof addr);**

**if (connect\_res == -1) {**

**perror("connect failed");**

**exit(EXIT\_FAILURE);**

**}**

**ssize\_t nread;**

**signal(SIGINT, killing\_handler);**

**char buf[256];**

**for (int i = 0; i < WRITERS\_SIZE; ++i) {**

**if (fork() == 0) {**

**while (1) {**

**if (killed) {**

**printf("Клиент писателя закрыт\n");**

**close(fd\_socket);**

**exit(0);**

**}**

**// отправляет номер писателя**

**sprintf(buf, "%d", i);**

**write(fd\_socket, buf, 256);**

**// отправляет номер книги**

**sprintf(buf, "%d", rand() % LIBRARY\_SIZE);**

**write(fd\_socket, buf, 256);**

**sleep(rand() % 2 + 5);**

**}**

**exit(0);**

**}**

**}**

**return 0;**

**}**

## Файл “writers.c”

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

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <signal.h>**

**#define PORT 8888**

**sig\_atomic\_t killed = 0;**

**void killing\_handler(int sig) {**

**killed = 1;**

**}**

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

**int fd\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (fd\_socket == -1) {**

**perror("socket failed");**

**exit(EXIT\_FAILURE);**

**}**

**struct sockaddr\_in addr = { 0 };**

**if (argc == 2) {**

**addr.sin\_addr.s\_addr = inet\_addr(argv[0]);**

**}**

**addr.sin\_family = AF\_INET;**

**addr.sin\_port = htons(PORT);**

**if (argc == 2) {**

**int port;**

**sscanf(argv[1], "%d", &port);**

**addr.sin\_port = htons(port);**

**}**

**int connect\_res = connect(fd\_socket, (struct sockaddr\*)&addr, sizeof addr);**

**if (connect\_res == -1) {**

**perror("connect failed");**

**exit(EXIT\_FAILURE);**

**}**

**ssize\_t nread;**

**signal(SIGINT, killing\_handler);**

**char buf[256];**

**while (1) {**

**if (killed) {**

**printf("Клиент дисплея закрыт\n");**

**write(STDOUT\_FILENO, buf, nread);**

**close(fd\_socket);**

**exit(0);**

**}**

**nread = read(fd\_socket, buf, 256);**

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

**perror("read failed");**

**exit(EXIT\_FAILURE);**

**}**

**if (nread == 0) {**

**printf("END OF FILE occured\n");**

**}**

**printf("%s\n", buf);**

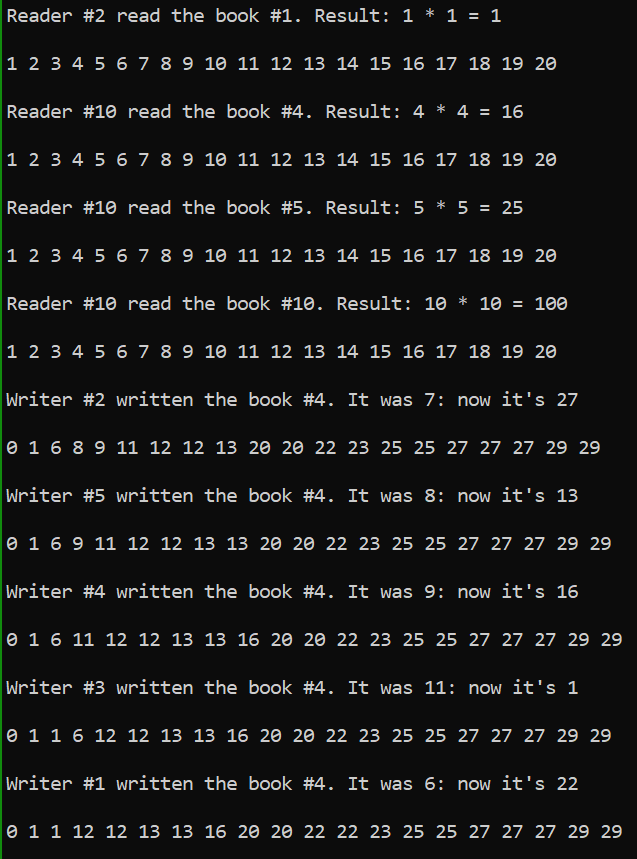
**}**

**return 0;**

**}**

## 

## Тесты



## 

## Выполненные критерии

**4-5 баллов:**

* Соблюдены общие требования к отчету.
* В отчете приведен сценарий решаемой задачи, поясняющий, каким образом исходные сущности и их поведение отображаются в серверы, клиенты, процессы и как осуществляется их взаимодействие.
* При запуске программ требуемые для их работы IP адреса и порты можно задавать в командной строке.
* Разработанное приложение работает как на одном компьютере так и в распределенном (сетевом) режиме на нескольких компьютерах, по которым можно будет разнести серверы и клиентов.

**6-7 баллов:**

* В дополнение к программе на предыдущую оценку разработана отдельная клиентская программа, подключаемая к серверу, которая предназначена для отображения комплексной информации о выполнении приложения в целом.

**8 баллов:**

* В дополнение к предыдущей программе реализована возможность подключения множества клиентов, обеспечивающих отображение информации о работе приложения.