ИДЗ 3, Бабушкин Владимир Алексндрович, вариант 22, БПИ237

Задача о картинной галерее. Вахтер следит за тем, чтобы в картинной галерее одновременно было не более 25 посетителей. Для обозрения представлены 5 картин. Каждый посетитель слу- чайнопереходит от картины к картине, но если на желаемую кар- тину любуются более пяти посетителей, он стоит в стороне и ждет, пока число желающих увидеть эту картину не станет меньше. По- 20 сетитель покидает галерею по завершении осмотра всех картин. Каждый посетитель уникальный, то есть имеет свой номер (на- пример, PID). В галерею также пытаются постоянно зайти новые посетители, которые ожидают своей очереди и разрешения от вах- тера, если та заполнена.

Создать клиент–серверное приложение, моделирующее од- нодневную работу картинной галереи (например, можно ограничить числом посетителей от 100 до 300).

ч Вахтер — сервер. Картинная галерея — клиент. Приходящиие посетители реализуются общим клиентом. Время нахождения возле картины для каждого посетителя является случайной ве- личиной в некотором диапазоне

Вариант 4–5

Архитектура системы

Сервер (`server.c`)

Функционал:

1. Управляет подключениями клиентов через `select()`
2. Обрабатывает команды:
   - `ENTER` - вход в галерею
   - `VIEW` - просмотр картины
   - `LEAVE_PAINTING` - уход от картины
   - `EXIT` - выход из галереи
3. Отслеживает:
   - Общее количество посетителей (`current_visitors`)
   - Количество зрителей у каждой картины (`painting_counts[5]`)

Ключевые структуры:

```
typedef struct {
    int fd;              // Дескриптор сокета
    int visitor_id;      // ID посетителя
    int current_painting // Текущая картина (-1 если не смотрит)
} client_t;
```

Клиент (`client.c`)

Логика работы:

1. Подключается к серверу с повторными попытками
2. Последовательно:
    - Запрашивает вход (`ENTER`)
    - Случайно выбирает картины для просмотра (`VIEW`)
    - Проводит у картины 1-3 секунды (случайно)
    - Уходит от картины (`LEAVE_PAINTING`)
    - Выходит после просмотра всех 5 картин (`EXIT`)

Особенности:

- Таймауты на операции (`READ_TIMEOUT_SEC`)
- Повторные попытки при ошибках (`MAX_RETRIES`)

Сценарий взаимодействия

1. **Запуск сервера:**

```
./server 8080
```

2. **Запуск клиентов (в отдельных терминалах):** при помощи написаноого bash-скрипта `run.sh`

```
#!/bin/bash

SERVER_IP="127.0.0.1"
SERVER_PORT="8080"
NUM_VISITORS="100"

echo "Запуск $NUM_VISITORS посетителей..."

for ((i=1; i<=$NUM_VISITORS; i++))
do
    ./client $SERVER_IP $SERVER_PORT $i &
done

echo "Все посетители запущены"
wait
echo "Все посетители завершили работу"
```

3. **Пример диалога:**

клиенты:

```
Visitor 2 received: VIEW_OK 2 0
Visitor 1 received: LEFT_PAINTING 1
```

```
Visitor 1 received: VIEW_WAIT 1 1
Visitor 9 received: LEFT_PAINTING 9
Visitor 8 received: LEFT_PAINTING 8
Visitor 10 received: LEFT_PAINTING 10
Visitor 8 received: VIEW_OK 8 3
Visitor 10 received: VIEW_WAIT 10 2
Visitor 9 received: VIEW_OK 9 2
Visitor 21 received: LEFT_PAINTING 21
Visitor 21 received: VIEW_OK 21 1
Visitor 20 received: LEFT_PAINTING 20
Visitor 20 received: VIEW_WAIT 20 1
Visitor 24 received: LEFT_PAINTING 24
Visitor 24 received: VIEW_OK 24 2
```

сервер:

```
Received from 93: 93 ENTER -1
Received from 5: 5 LEAVE_PAINTING 4
Received from 5: 5 VIEW 2
Received from 4: 4 VIEW 4
Received from 2: 2 VIEW 0
Received from 1: 1 LEAVE_PAINTING 3
Received from 1: 1 VIEW 1
Received from 8: 8 LEAVE_PAINTING 0
Received from 9: 9 LEAVE_PAINTING 0
Received from 10: 10 LEAVE_PAINTING 1
Received from 8: 8 VIEW 3
Received from 9: 9 VIEW 2
Received from 10: 10 VIEW 2
Received from 21: 21 LEAVE_PAINTING 3
Received from 21: 21 VIEW 1
```

Обработка ошибок

1. **На сервере:**
    - Контроль лимитов (MAX_VISITORS, MAX_PAINTING_VISITORS)
    - Валидация формата сообщений
    - Автоматическая очистка отключившихся клиентов
2. **На клиенте:**
    - Повторные подключения при сбоях
    - Таймауты ожидания ответа
    - Корректное завершение сеанса

Пример вывода

Сервер:

```
Сервер запущен на порту 8080
Получено от 101: 101 ENTER -1
Получено от 101: 101 VIEW 2
Получено от 102: 102 ENTER -1
```

Клиент 101:

```
Visitor 101 received: ENTER_OK 101
Visitor 101 received: VIEW_OK 101 2
Visitor 101 received: LEFT_PAINTING 101
```

вариант 6-7

Добавленные компоненты

1. Модуль мониторинга (`monitor.c`)

Ключевые особенности:

- Использует библиотеку `ncurses` (нашел в интернете такую удобную) для интерактивного вывода
- Подключается к серверу как специальный клиент-наблюдатель
- Получает и визуализирует общее состояние системы:
  - Общее количество посетителей
  - Загрузку по каждой картине
  - Ограничения системы (25/25, 5/5)

Реализация:

```c
// Инициализация ncurses
initscr();
cbreak();
noecho();
keypad(stdscr, TRUE);
nodelay(stdscr, TRUE);

// Основной цикл обновления
while (1) {
    print_gallery_status(visitors, paintings);
    if (getch() == 'q') break;
    napms(100);
}
```

2. Модификации сервера

Новые функции:

1. Поддержка клиентов-мониторов:

```
if (strcmp(buffer, "MONITOR") == 0) {
    clients[client_idx].is_monitor = 1;
}
```

2. Широковещательная рассылка состояния:

```
void broadcast_status() {
    char status[BUFFER_SIZE];
    snprintf(status, "..."); // Формирование строки состояния
    for (все мониторы) send(...);
}
```

Пример работы системы

Запуск:

точно также, только теперь еще

```
./monitor 127.0.0.1 8080
```

Вывод монитора:

```
=== Картинная галерея ===

Общее количество посетителей: 25/25

Посетители у картин:
Картина 1: 5/5 посетителей
Картина 2: 5/5 посетителей
Картина 3: 5/5 посетителей
Картина 4: 2/5 посетителей
Картина 5: 4/5 посетителей

Нажмите q для выхода
```

8

Уже реализованные возможности

В предыдущей версии системы были реализованы все необходимые механизмы для работы с множеством клиентов-наблюдателей:

1. **Динамическое подключение/отключение мониторов**
    - Сервер поддерживает неограниченное количество мониторов
    - Каждый монитор может независимо подключаться и отключаться

```
// Сервер принимает новые подключения мониторов
if (strcmp(buffer, "MONITOR") == 0) {
    clients[client_idx].is_monitor = 1;
    broadcast_status(); // Сразу отправляем текущее состояние
}
```

2. **Широковещательный механизм рассылки**
    - Сервер автоматически рассылает состояние всем подключенным мониторам

```
void broadcast_status() {
    char status[BUFFER_SIZE];
    snprintf(status, "...");
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].is_monitor) {
            send(clients[i].fd, status, strlen(status), 0);
        }
    }
}
```

3. **Устойчивость к разрывам соединений**
    - При отключении монитора сервер продолжает работать
    - Новые мониторы получают актуальное состояние системы

```
// Обработка отключения клиента
if (bytes_read <= 0) {
    close(clients[i].fd);
    clients[i].fd = -1;
    // Не влияет на работу других клиентов
}
```

Демонстрация работы

Сценарий использования: точно также, только теперь еще

```
./monitor 127.0.0.1 8080
```

и то же самое в другом окне

Результат: В обоих окнах выводится одна и таже информация

9

Внесенные изменения в клиентскую часть

1. Обработка сигналов

Добавлен механизм корректного завершения работы клиента при получении сигналов:

```
void cleanup(int sig) {
    if (g_sock != -1) {
        char buffer[BUFFER_SIZE];
        snprintf(buffer, BUFFER_SIZE, "%d EXIT -1", g_visitor_id);
        send(g_sock, buffer, strlen(buffer), 0);
        close(g_sock);
        printf("\nVisitor %d: Cleanup complete. Exiting.\n",
g_visitor_id);
    }
    exit(EXIT_SUCCESS);
}

void setup_signal_handlers() {
    struct sigaction sa;
    sa.sa_handler = cleanup;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    if (sigaction(SIGINT, &sa, NULL) == -1) {
        perror("Failed to set up SIGINT handler");
        exit(EXIT_FAILURE);
    }
    if (sigaction(SIGTERM, &sa, NULL) == -1) {
        perror("Failed to set up SIGTERM handler");
        exit(EXIT_FAILURE);
    }
}
```

2. Глобальные переменные

Добавлены глобальные переменные для доступа из обработчика сигналов:

```
static int g_sock = -1;
static int g_visitor_id = -1;
```

3. Модификация main()

Инициализация обработчиков сигналов в начале main():

```
setup_signal_handlers();
```

Результаты работы

1. При нажатии Ctrl+C в терминале клиента:
    - Клиент отправляет серверу сообщение EXIT
    - Сервер корректно обрабатывает выход посетителя
    - Обновляет счетчики и рассылает новое состояние мониторам

10

Внесенные изменения в серверную часть

1. Обработка сигналов на сервере

Добавлен механизм корректного завершения работы сервера:

```
volatile sig_atomic_t stop_server = 0;

void notify_shutdown() {
    const char *msg = "SERVER_SHUTDOWN";
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].fd != -1) {
            send(clients[i].fd, msg, strlen(msg), 0);
            close(clients[i].fd);
        }
    }
}

void handle_signal(int sig) {
    if (stop_server) return;
    stop_server = 1;
    notify_shutdown();
    exit(EXIT_SUCCESS);
}

void setup_signal_handlers() {
    struct sigaction sa;
    sa.sa_handler = handle_signal;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGTERM, &sa, NULL);
    signal(SIGPIPE, SIG_IGN);
}
```

2. Модификация главного цикла сервера

```
while (!stop_server) {
    // Основная логика работы сервера
}
```

Внесенные изменения в клиентскую часть

1. Обработка сообщения о shutdown

```
int process_server_response(char* response) {
    if (strstr(response, "SERVER_SHUTDOWN") != NULL) {
        return -1;
    }
    return 0;
}
```

2. Глобальные флаги и структуры

```
volatile sig_atomic_t g_shutdown = 0;
client_data_t g_client;

void handle_signal(int sig) {
    g_shutdown = 1;
}
```

Результаты работы

1. При завершении сервера (Ctrl+C или SIGTERM):
    - Сервер рассылает всем клиентам сообщение SERVER_SHUTDOWN
    - Корректно закрывает все соединения
    - Завершает свою работу
2. Клиенты при получении SERVER_SHUTDOWN:
    - Немедленно завершают работу
    - Освобождают ресурсы

```
if (process_server_response(response) == -1) {
    return -1; // Завершение работы клиента
}
```