Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Недосеков Иван Дмитриевич
Группа: М8О-206Б-19
Преподаватель: Соколов Андрей Алексеевич

Дата: _____
Оценка: _____
Подпись: _____

Москва, 2020

Цель проекта

Целью курсового проекта я выбрал межсетевые крестики нолики. Сетевую часть было решено написать на ZMQ. Сетевое взаимодействие через TCP протокол. Клиент и сервер связываются по паттерну Peer to Peer. Программа запущенная как сервер ждет подключения клиента. Можно задавать размер поля игры и общаться в чате. Интерфейс написан на ncurses

Общие сведения о программе

Программа при запуске требует 2 аргумента : роль игрока (клиент или сервер) и размер поля(пое квадратное поэтому задается 1 число). Третим необязательным параметром является IP адрес на котором создавать сервер/ к которому коннектиться. В каждом клиенте игры 2 потока: в одном крутится роутер, который занимается отправкой и получением сообщений по сети; во втором потоке производится отрисовка интерфейса и взаимодействие с ядром игры. Взаимодействие между потоками происходит через ZMQ через протокол ipc.

Роутер в бесконечном цикле проверяет 2 сокета: один с сети, другой от интерфейса. Пересылает пакеты, если приходит завершающий, то роутер выключается.

Интерфейс в бесконечном цикле проверяет пришло ли какое то задание на отрисовку от оппонента, далее проверяет нажата ли клавиша, если нажата, то по контексту выполняет команду.

Команды могут быть:

- *q* — выход (сдаться);
- *+* — передвинуться на следущее окно;
- *-* — передвинуться на предыдущее окно;
- *'_' (Пробел)* — выполнить действие в окне (начать писать сообщение в чат или поставить свою фигуру);

Ядро игры

Состоянии игры храниться в двумерном массиве чаров, при каждой постановке фигуры проверяется привел ли этот ход к победе. Проверка осуществляется с помощью задания направлений прямой проходящей через позицию, таких прямых 4.

```
bool check_case(core* c, int pos_x, int pos_y, int dx, int dy) {
   int count_right = 0;
   int curpos_x = pos_x, curpos_y = pos_y;
   while (curpos_x >= 0 && curpos_y >= 0 && curpos_x < c->size && curpos_y <
    ↪  c->size) {
      if (c->board[curpos_x][curpos_y] == c->my_side) {
         count_right++;
      } else {
         return false;
      }
      curpos_x += dx;
      curpos_y += dy;
   }
   curpos_x = pos_x - dx; curpos_y = pos_y - dy;
   while (curpos_x < c->size && curpos_y < c->size && curpos_x >= 0 &&
    ↪  curpos_y >= 0) {
      if (c->board[curpos_x][curpos_y] == c->my_side) {
         count_right++;
      } else {
```

```
            return false;
        }
        curpos_x -= dx;
        curpos_y -= dy;
    }
    return count_right == c->size;
}


bool  check_board(core* c, int x, int y) {
    if (check_case(c, x, y, -1, 0)) {
        return true;
    }
    if (check_case(c, x, y, -1, -1)) {
        return true;
    }
    if (check_case(c, x, y, 0, -1)) {
        return true;
    }
    if (check_case(c, x, y, 1, -1)) {
        return true;
    }
    return false;
}
```

При удачной постановки о состоянии игры уведомляется оппонент.

Роутер

Слушает 2 сокета и дополняет служебной информацией при необходимости.

```
void router_module(void* information) {
    ports my_ports;
    player_info* info = (player_info*)information;
    ports_init(&my_ports, info);
    send_first_information(&my_ports, info);

    while (true) {
        zmq_msg_t task;
        zmq_msg_init_size(&task, sizeof(message));
        int sizeof_message = zmq_msg_recv(&task, my_ports.opponent,
        ↪  ZMQ_DONTWAIT);
        if (sizeof_message > 0) {
            zmq_msg_send(&task, my_ports.tasks, 0);
            zmq_msg_close(&task);
```

```
            sleep(1);

            zmq_msg_t ans;
            zmq_msg_init(&ans);
            zmq_msg_send(&ans, my_ports.opponent, 0);
            zmq_msg_close(&ans);
         }
         zmq_msg_init_size(&task, sizeof(message));
         sizeof_message = zmq_msg_recv(&task, my_ports.to_send, ZMQ_DONTWAIT);
         if (sizeof_message > 0) {
            if (((message*)zmq_msg_data(&task))->type == QUIT) {
               zmq_msg_send(&task, my_ports.opponent, 0);
               break;

            } else {
               zmq_msg_send(&task, my_ports.opponent, 0);
            }
            zmq_msg_close(&task);
            zmq_msg_t ans;
            sleep(1);
            zmq_msg_init(&ans);
            zmq_msg_recv(&ans, my_ports.opponent, 0);
            zmq_msg_close(&ans);
         }
         sleep(1);

      }
      router_deinit(&my_ports);
   }
```

Интерфейс

```
void parse(message* m, core* c, parts* p, player_info* info) {
   if (m->type == TURN) {
      assert( m->size == c->size );
      char ch;
      int x, y;
      sscanf(m->data, "%c%d%d", &ch, &x, &y);
      core_turn(c, x * c->size + y, p, ch);
   } else if (m->type == OPPONENT_WIN) {
      info->how_game_ended = I_LOSE;
      system_message(p, "opponent win");
   } else if (m->type == CHAT) {
```

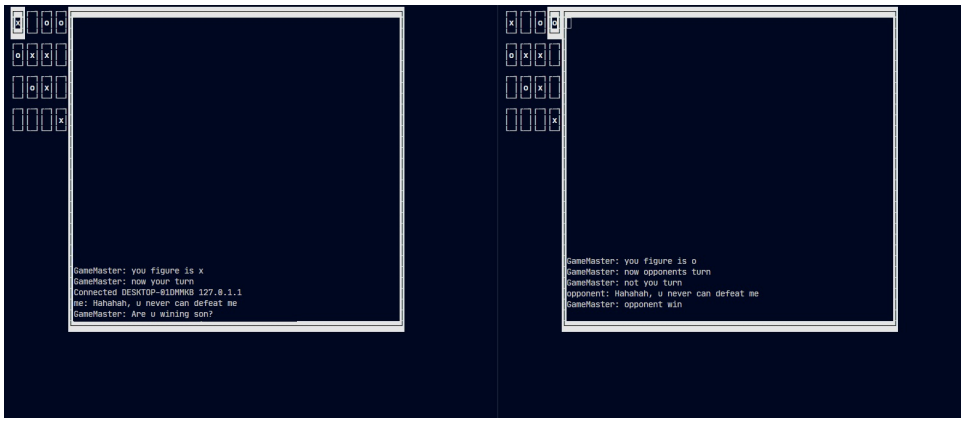
```
        char formatted[BUF_SIZE];
        sprintf(formatted, "opponent: %s", m->data);
        chat_push(p, formatted);
    } else if (m->type == QUIT) {
        system_message(p, "opponent exited from game");
    } else if (m->type == CONNECT) {
        chat_connected(p, m->data);
    }

}

void check_task(core* c, parts* p, player_info* info) {
    zmq_msg_t to_recv;
    zmq_msg_init_size(&to_recv, sizeof(message));
    int size = zmq_msg_recv(&to_recv, p->TASKS, ZMQ_DONTWAIT);
    if (size > 0) {
        message recv;
        memcpy(&recv, zmq_msg_data(&to_recv), sizeof(message));
        parse(&recv, c, p, info);
        zmq_msg_close(&to_recv);
    }
}


void chat_enable(parts* p) {
    p->chat_is_enabled = true;
    char buf[BUF_SIZE];
    char formatted[BUF_SIZE];
    echo();
    curs_set(1);
    assert(mvwgetnstr(p->CHAT, 28, 1, buf, BUF_SIZE) != ERR);
    sprintf(formatted, "me: %s", buf);
    chat_push(p, formatted);
    noecho();
    curs_set(0);
    send_chat_message(p, buf);
    p->chat_is_enabled = false;
}

void first_message(parts* p, player_info* info) {
    char fm[BUF_SIZE];
    sprintf(fm, "you figure is %c", info->my_figure);
    system_message(p, fm);
    if (info->is_my_turn) {
        sprintf(fm, "now your turn");
```

```
    } else {
        sprintf(fm, "now opponents turn");
    }
    system_message(p, fm);
}

void interface(void* information) {
    player_info* info = information;
    parts this_interface;
    core my_core;
    interface_initialise(&this_interface, &my_core, info);
    int key = '\0';
    int sq = 0;
    int size_of_board = (info->size) * (info->size);

    highlight_square(this_interface.BOARD[sq], &this_interface);
    first_message(&this_interface, info);
    do {
        check_task(&my_core, &this_interface, info);
        if (!kbhit()) {
            continue;
        }
        key = getch();
        if (key == '+' && sq < size_of_board) {
            draw_square(this_interface.BOARD[sq], 0, &this_interface);
            ++sq;
            if (sq == size_of_board) {
                highlight_square(this_interface.CHAT, &this_interface);
            } else {
                highlight_square(this_interface.BOARD[sq], &this_interface);
            }
        } else if (key == ' ') {
            if (sq == size_of_board) {
                chat_enable(&this_interface);
                continue;
            }
            if (!my_core.is_my_turn) {
                system_message(&this_interface, "not you turn");
                continue;
            }
            if (core_turn(&my_core, sq, &this_interface, my_core.my_side)) {
                draw_square(this_interface.BOARD[sq], my_core.my_side,
                ↪  &this_interface);
                highlight_square(this_interface.BOARD[sq], &this_interface);
                if (my_core.win) {
```

```
                info->how_game_ended = I_WIN;
                system_message(&this_interface, "Are u wining son?");
            }
        } else {
            system_message(&this_interface, "you cant set figure there");
        }
    } else if (key == '-' && sq > 0) {
        if (sq == size_of_board) {
            draw_square(this_interface.CHAT, 0, &this_interface);
        } else {
            draw_square(this_interface.BOARD[sq], 0, &this_interface);
        }
        --sq;
        highlight_square(this_interface.BOARD[sq], &this_interface);
    }
  } while ((key != 'q') && (key != 'Q'));
  deinitialize(&this_interface, info, &my_core);
}
```

Пример работы

Создано поле 4 на 4:



В чате можно общаться, так же в нем отображается лог и сообщения игры для пользователя. Также была проверено что сетевая часть действительно работает. Приложение было собрано на *termux* и запущено как клиент, сервер был в локальной сети на Linux Monjaro.

GitHub

Ссылка на репозиторий GitHub с проектом:
https://github.com/GrozniyToaster/os_kp

Вывод

Было интересно разрабатывать курсовой проект, во время продумывания взаимодействий потоков и клиентов игры. Когда приложение, которое до этого работало только на локалхосте и без лишних доработок запускается на двух совершенно разных машинах (телефоне и ноутбуке) испытываешь огромное удовольствие. Проект дал практику в связывании нескольких, проходимых на лабораторных, темах: потоки, очереди сообщений и работа с файлами. Также я познакомился с библиотекой ncurses, для оформления интерактивного интерфейса.