Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6-8 по курсу «Операционные системы»

Студент: Рылов Александр Дмитриевич
Группа: М8О-207Б-21
Вариант: 24
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Brokiloene/os

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:
- Управлении серверами сообщений (№2)
- Применение отложенных вычислений (№4)
- Интеграция программных систем друг с другом (№3)

Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать два вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

• Создание нового вычислительного узла;

• Удаление существующего вычислительного узла;

• Исполнение команды на вычислительном узле;

• Проверка доступности вычислительного узла.

Задание варианта

Вариант 24.

Дерево общего вида

Исполнение команды — поиск подстроки в строке.

Команда проверки — проверка доступности всех узлов.

Общие сведения о программе

Программа распределительного узла компилируется из файла main.c, программа вычислительного узла компилируется из файла node.c. В программе используется библиотека для работы с сервером сообщений ZeroMQ. В программе используются следующие системные вызовы:

fork — создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
execl — используется для выполнения другой программы. Эта другая программа, называемая процессом-потомком (child process), загружается поверх программы, содержащей вызов exec. Имя файла, содержащего процесс-потомок, задано с помощью первого аргумента. Какие-либо аргументы, передаваемые процессу-потомку, задаются либо с помощью параметров от arg0 до argN, либо с помощью массива arg[].
Также были использованы следующие вызовы из библиотеки ZMQ:
zmq_ctx_new — создает новый контекст ZMQ.
zmq_connect — создает входящее соединение на сокет.
zmq_disconnect — отсоединяет сокет от заданного endpoint'a.
zmq_socket — создает ZMQ сокет.
zmq_close — закрывает ZMQ сокет.
zmq_ctx_destroy — уничтожает контекст ZMQ.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с ZMQ.

2. Проработать принцип общения между клиентскими узлами и между первым клиентом и сервером и алгоритм выполнения команд клиентами.

3. Реализовать необходимые функции-обертки над вызовами функций библиотеки ZMQ.

4. Написать программу сервера и клиента

Исходный код

manage_node.c

```
#include "list.h"
#include "lab6_utils.h"
```

```
#include <zmq.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //fork
#include <string.h> //strcmp
#include <sys/wait.h>


int main(int argc, char const *argv[])
{
    void *context = zmq_ctx_new();
    void *requester = zmq_socket(context, ZMQ_REQ);
    check_null(requester, "requester error");

    List *childs = list_create();
    check_null(childs, "list_create error");

    char cmd[10];
    int c;
    printf("> ");
    while((c = getchar()) != EOF) {
        ungetc(c, stdin);

        //printf("[mn]: list: ");
        //print_list(childs);
        printf("> ");

        scanf("%s", cmd);
        if (strcmp(cmd, "create") == 0) {
                                                //print_list(childs);

            int id, parent_id;
            scanf("%d %d", &id, &parent_id);
            id += PORT;
```

```
        if (parent_id == -1) {
            if (list_find(childs, id) == 1) {
                printf("bad child id\n");
                continue;
            }

            int pid = fork();
            check_neg_one(pid, "fork error");

            if (pid == 0) {
                                                zmq_close(requester);
                                                                zmq_ctx_destroy(context);
                                                                list_destroy(childs);

                char id_str[8];
                memset(id_str, 0, 8);
                sprintf(id_str, "%d", id);
                char *argvc[] = {"computing_node", id_str, NULL};
                check_neg_one(execv("computing_node", argvc), "execv computing_node error");
            }

            //id -= PORT;
            //printf("[mn59]: %s %d %d\n", cmd, id, parent_id);
            sleep(1); // если создать/удалить/снова создать процесс, ping не сработает (???)
            //printf("ping= %d\n", ping(id));
            if (ping(id) == 1) {
                                                        push_back(childs, id);
                                                        printf("computing_node № %d with pid [%d] has been
created\n", id, pid);
                                                    }
        }

        else {
                                                parent_id += PORT;
            // if (list_find(childs, parent_id) == 0 || list_find(childs, id) == 1) {
            //  printf("bad parent/child id\n");
```

```
//  continue;
// }

                                        //printf("I am here\n");

if (list_find(childs, parent_id) == 1 && ping(parent_id) == 1) {
                                        void *requester2 = zmq_socket(context, ZMQ_REQ);
   Message m = {"create", parent_id, id, ""};

   //printf("childid=%d, parent_id=%d\n", id, parent_id);

   char address[32];
   client_address_gen(parent_id, address);
   //printf("address=%s", address);
   check_neg_one(zmq_connect(requester2, address), "zmq_connect error");
   send_msg(&m, requester2);
   recv_msg(&m, requester2);

   zmq_close(requester2);

   //push_back(childs, m.id);

   if (strcmp(m.cmd, "bad") == 0) {
      printf("%s: error with creating [%d] by [%d]\n", m.cmd, m.id, m.num);
   } else {
      printf("%s: [%d] was created by [%d]\n", m.cmd, m.id, parent_id);
   }
}
else {
                                        //printf("I am here\n");
                                         // break;
   int cur_id = echo(childs, parent_id, requester);

   //printf("[mn]: cur_id = %d\n", cur_id);

   if (cur_id == -1 || ping(cur_id) != 1) {
```

```
            printf("bad id\n");
            continue;
        }

        Message m = {"create", parent_id, id, ""};

        char address[32];
        client_address_gen(cur_id, address);
        check_neg_one(zmq_connect(requester, address), "zmq_connect error");
        send_msg(&m, requester);
        recv_msg(&m, requester);

        //printf("[mn110]: %s %d %d\n", m.cmd, m.id, m.num);

        if (strcmp(m.cmd, "bad") == 0) {
            printf("%s: error with creating [%d] by [%d]\n", m.cmd, m.id, m.num);
        }
        else if (strcmp(m.cmd, "ok") == 0) {
            printf("%s: [%d] was created by [%d] via [%d]\n", m.cmd, m.id, parent_id, cur_id);
        }

      }

    }
  } else if (strcmp(cmd, "exec") == 0) {
    int id;
    scanf("%d", &id);
    id += PORT;
    getchar();  // scanf сам не считывает '\n'

    char *pattern = (char *) malloc(sizeof(char) * 1024), *text = NULL;
    size_t len = 0;
    int pattern_size = getline(&pattern, &len, stdin);
    int text_size    = getline(&text,    &len, stdin);

    pattern_size -= 1;    // getline выдает количество считанных символов
```

```
        pattern[pattern_size] = '#';  // " для char *, ' для char
        char *all    = strcat(pattern, text);

        Message m = {"exec", pattern_size, id, ""};
        memset(m.str, 0, 1024);
        memcpy(m.str, all, strlen(all));

        //printf("[mn]: exec id=%d, lf=%d\n", id, list_find(childs, id));


        if (list_find(childs, id) == 1 && ping(id) == 1) {
                                        void *requester2 = zmq_socket(context, ZMQ_REQ);
                                        //printf("145\n");
                                        char address[32];
                                                client_address_gen(id, address);
                                                printf("[mn]: address=%s\n", address);
                                                //check_neg_one(zmq_connect(requester2, address),
"zmq_connect error");
                                                //printf("152\n");
                                                send_msg(&m, requester2);
                                                //printf("154\n");
                                                recv_msg(&m, requester2);
                                                //printf("156\n");

                                                  printf("%s:\n%s \nproduced by [%d]\n", m.cmd, m.str,
m.id);
                                                  zmq_close(requester2);
        } else {
                                        //printf("160\n");
                                        void *echo_requester = zmq_socket(context, ZMQ_REQ);
                                        int cur_id = echo(childs, id, echo_requester);
                                        zmq_close(echo_requester);

          //printf("[mn]: cur_id = %d\n", cur_id);

          if (cur_id == -1 || ping(cur_id) == -1) {
```

```
            printf("bad id\n");
            free(pattern);
                                                    free(text);
            continue;
        }

        char address[32];
        client_address_gen(cur_id, address);
        //printf("[mn]: address=%s\n", address);
        check_neg_one(zmq_connect(requester, address), "zmq_connect error");
        send_msg(&m, requester);
        recv_msg(&m, requester);

        //printf("[mn110]: %s %d %d\n", m.cmd, m.id, m.num);

        if (strcmp(m.cmd, "bad") == 0) {
            printf("bad id\n");
            //printf("180\n");
        }
        else if (strcmp(m.cmd, "ok") == 0) {
            printf("%s:\n%s \nproduced by [%d]\n", m.cmd, m.str, m.id);
        }
    }
                                        free(pattern);
                                        free(text);

}

else if (strcmp(cmd, "ping") == 0) {
    int id;
    scanf("%d", &id);
    id += PORT;
    printf("[%d]: %d\n", id, ping(id));
}
else if (strcmp(cmd, "remove") == 0) {
```

```
        int id;
        scanf("%d", &id);
        id += PORT;

        Message m = {"remove", 0, id, ""};

        //printf("[mn]: %s %d %d\n", m.cmd, m.id, m.num);

        if (list_find(childs, id) == 1 && ping(id) == 1) {

                                        void *requester2 = zmq_socket(context, ZMQ_REQ);


                                                char address[32];
                                                client_address_gen(id, address);
                                                check_neg_one(zmq_connect(requester2, address),
"zmq_connect error");
                                                send_msg(&m, requester2);
                                                recv_msg(&m, requester2);

                                                list_delete(childs, id);

                                                zmq_close(requester2);

                                                printf("%s: [%d] with pid %d has been removed\n",
m.cmd, m.id, m.num);
                                                wait(NULL);  // а то будет зомби

        } else {
                                        int cur_id = echo(childs, id, requester);

           //printf("[mn]: cur_id = %d\n", cur_id);

           if (cur_id == -1 || ping(cur_id) == -1) {
              printf("bad id\n");
              continue;
```

```
        }

        char address[32];
        client_address_gen(cur_id, address);
        check_neg_one(zmq_connect(requester, address), "zmq_connect error");
        send_msg(&m, requester);
        recv_msg(&m, requester);

        if (strcmp(m.cmd, "bad") == 0) {
              printf("%s: cannot remove %d", m.cmd, m.id);
        }
        else if (strcmp(m.cmd, "ok") == 0) {
           printf("%s: [%d] with pid %d has been removed via [%d]\n", m.cmd, m.id, m.num, cur_id);
        }

    }
}
else if (strcmp(cmd, "heartbit") == 0) {
                                        int delay;
    scanf("%d", &delay);


    for (int i = 0, sz = size(childs); i < sz; ++i) {
                                        int cur_id = list_get(childs, i);
                                        check_neg_one(cur_id, "get error");

                                        //printf("[mn]: delay=%d cur_id=%d\n", delay, cur_id);

                                        int answers = 0;
                                        for (int j = 0; j < 4; ++j) {
                                         if (ping(cur_id) == 1) {
                                                answers += 1;
                                         }
                                         usleep(delay * 1000);
                                        }
```

```
                                        if (answers == 4) {
                                                printf("[%d]: ready\n", cur_id);

                                                Message m = {"heartbit", delay, 0, ""};

                                                char address[32];
                                                client_address_gen(cur_id, address);


                                        check_neg_one(zmq_connect(requester, address),
"zmq_connect error");

                                                send_msg(&m, requester);
                                                recv_msg(&m, requester);

                                                zmq_disconnect(requester, address);
                                        } else {
                                                printf("[%d]: bad\n", cur_id);
                                        }



        }
    }
  }

  for (int i = 0, sz = size(childs); i < sz; ++i) {
                                        void *requester2 = zmq_socket(context, ZMQ_REQ);
                                         int cur_id = list_get(childs, i);
                                         check_neg_one(cur_id, "get error");
                                         if (ping(cur_id) == 1) {

                                                Message m2 = {"remove", 0, cur_id, ""};

                                                char address[32];
                                                client_address_gen(cur_id, address);
```

```
                                        check_neg_one(zmq_connect(requester2, address),
"zmq_connect error");

                                        send_msg(&m2, requester2);
                                        recv_msg(&m2, requester2);

                                        zmq_close(requester2);
                                    }
                                    }

   zmq_close(requester);
   zmq_ctx_destroy(context);
   list_destroy(childs);

   return 0;
}
```

computing_node.c

```
#include "list.h"
#include "lab6_utils.h"

#include <zmq.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //fork
#include <string.h> //strcmp
#include <sys/wait.h>



int main(int argc, char const *argv[])
{
                                    if (argc < 2) {
                                     perror("argc < 2");
```

```
                                        exit(EXIT_FAILURE);
                                        }
                                        int my_id = atoi(argv[1]);

                                        //printf("%d\n", my_id);

                                        List *childs = list_create();
                                        check_null(childs, "list_create error");

                                        void *context = zmq_ctx_new();
    void *responder = zmq_socket(context, ZMQ_REP);

    char address[32];
                                        memset(address, 0, 32);
                                        memcpy(address, SERVER_PATTERN, sizeof(SERVER_PATTERN));
                                        strcat(address, argv[1]);

    check_neg_one(zmq_bind(responder, address), "zmq bind error");

    //printf("[%d]: bound to %s\n", my_id, address);

    while(1) {
                                        Message m;
                                        recv_msg(&m, responder);
                                        //printf("[%d]: %s %d %d\n", my_id, m.cmd, m.id, m.num);

      if (strcmp(m.cmd, "create") == 0) {
                                        if (m.num == my_id) {
                                         if (list_find(childs, m.id) == 1 || my_id == m.id) {

                                                //printf("I am here\n");
                                                //printf("[%d]: %s %d %d\n", my_id, m.cmd, m.id,
m.num);

                                                                memcpy(&m.cmd, "bad", sizeof(char) *
10);
15
```

```
                send_msg(&m, responder);
                }
                else {
                int pid = fork();
                        check_neg_one(pid, "fork error");

                        if (pid == 0) {
                                zmq_close(responder);
                        zmq_ctx_destroy(context);
                        list_destroy(childs);

                                char id_str[8];
                                memset(id_str, 0, 8);
                                sprintf(id_str, "%d", m.id);
                                char *argvc[] =
{"computing_node", id_str, NULL};

                        check_neg_one(execv("computing_node", argvc), "execv
computing_node error");
                                }
                                sleep(1);

                                memcpy(&m.cmd, "ok",
sizeof(char) * 10);
                                m.num = pid;
                        send_msg(&m, responder);

                        push_back(childs, m.id);
                }
                } else {
                        //printf("[%d]: %s %d %d\n", my_id, m.cmd,
m.id, m.num);
                        if (list_find(childs, m.num) == 1 &&
ping(m.num) == 1) {
                                Message m2 = {"create", m.num, m.id,
""};
```

```
                                        //printf("%s %d %d\n", m2.cmd, m2.id,
m2.num);

                                        void *requester = zmq_socket(context,
ZMQ_REQ);

                                        char address[32];
                                        client_address_gen(m2.num, address);

                                        // printf("address=%s", address);
                                        // printf("I am here\n");
                                        // break;


                                check_neg_one(zmq_connect(requester, address),
"zmq_connect error");

                                        send_msg(&m2, requester);
                                recv_msg(&m2, requester);
                                zmq_close(requester);

                                //memcpy(&m2.cmd, "ok", sizeof(char) * 10);
                                send_msg(&m2, responder);


                                //push_back(childs, m.id);
                                //printf("%s: [%d] was created by [%d]\n",
m.cmd, m.id, parent_id);
                                }
                                else {
                                        void *requester = zmq_socket(context,
ZMQ_REQ);

                                        int cur_id = echo(childs, m.num,
requester);
                                        if (cur_id == -1 || ping(cur_id) == -1) {
```

```
                                printf("bad_id\n");
                                memcpy(&m.cmd, "bad",
sizeof(char) * 10);
                                send_msg(&m, responder);
                                continue;
                        }

                        //printf("[%d]: list:\n", my_id);
        //print_list(childs);
        //printf("[%d]113: cur_id = %d\n", my_id, cur_id);

                        Message m2 = {"create", m.num, m.id,
""};
                        //printf("[%d]: %s %d %d\n", my_id,
m2.cmd, m2.id, m2.num);

                        char address[32];
                        client_address_gen(cur_id, address);
                        //printf("[%d]: address=%s", my_id,
address);

                        //sleep(30);



                check_neg_one(zmq_connect(requester, address),
"zmq_connect error");

                        send_msg(&m2, requester);
                recv_msg(&m2, requester);
                zmq_close(requester);

                send_msg(&m2, responder);


                //push_back(childs, m.id);
```

```
                                        //printf("%s: [%d] was created by [%d] via
[%d]\n", m.cmd, m.id, parent_id, cur_id);
                                        }
                                    }

    }
    else if (strcmp(m.cmd, "exec") == 0) {
                                //printf("[%d]: %s %d %d %s\n", my_id, m.cmd, m.id, m.num,
m.str);
                                //print_list(childs);
                                if (m.id == my_id) {
                                    memcpy(&m.cmd, "ok", sizeof(char) * 10);
                                    find_substrings(&m);
                                    send_msg(&m, responder);
                                } else if (list_find(childs, m.id) == 1 && ping(m.id) == 1) {
                                    void *requester = zmq_socket(context, ZMQ_REQ);
                                    char address[32];
                                        client_address_gen(m.id, address);
                                        //printf("[%d]150: address=%s", my_id,
address);

                                        check_neg_one(zmq_connect(requester,
address), "zmq_connect error");

                                        send_msg(&m, requester);
                                recv_msg(&m, requester);
                                zmq_close(requester);

                                send_msg(&m, responder);
                                }

                                else {
                                    void *requester = zmq_socket(context, ZMQ_REQ);

                                        int cur_id = echo(childs, m.id, requester);
                                        if (cur_id == -1 || ping(cur_id) == -1) {
```

```
                                                printf("bad_id\n");
                                                memcpy(&m.cmd, "bad", sizeof(char) *
10);
                                                send_msg(&m, responder);
                                                continue;
                                            }

                                            //printf("[%d]: list:\n", my_id);
        //print_list(childs);
        //printf("[%d]157: cur_id = %d\n", my_id, cur_id);

                                            //Message m2 = {"create", m.num, m.id, ""};
                                            //printf("[%d]: %s %d %d\n", my_id, m2.cmd,
m2.id, m2.num);

                                            char address[32];
                                            client_address_gen(cur_id, address);
                                            //printf("[%d]: address=%s", my_id, address);

                                            check_neg_one(zmq_connect(requester,
address), "zmq_connect error");

                                                send_msg(&m, requester);
                                    recv_msg(&m, requester);
                                    zmq_close(requester);

                                    send_msg(&m, responder);
                                        }
    }
    else if (strcmp(m.cmd, "remove") == 0) {

                                    //printf("[%d]: %s %d %d\n", my_id, m.cmd, m.id, m.num);

                                    if (m.id == my_id) {
                                     for (int i = 0, sz = size(childs); i < sz; ++i) {
                                                int cur_id = list_get(childs, i);
```

```
                                        check_neg_one(cur_id, "get error");

                                        if (ping(cur_id) == 1) {

                                                void *requester =
zmq_socket(context, ZMQ_REQ);

                                                Message m2 = {"remove", 0,
cur_id, ""};

                                                char address[32];
                                                client_address_gen(cur_id,
address);


                                check_neg_one(zmq_connect(requester, address),
"zmq_connect error");

                                                send_msg(&m2, requester);
                                        recv_msg(&m2, requester);
                                        zmq_close(requester);
                                        }
                                        }

                                 //printf("[%d]: %s %d %d\n", my_id, m.cmd, m.id, m.num);

                                 memcpy(&m.cmd, "ok", sizeof(char) * 10);
                                        m.num = getpid();
                                send_msg(&m, responder);
                                break;
                                } else if (list_find(childs, m.id) == 1 && ping(m.id) == 1) {

                                 Message m2 = {"remove", 0, m.id, ""};
                                        //printf("%s %d %d\n", m2.cmd, m2.id,
m2.num);
```

```
                                                    void *requester = zmq_socket(context,
ZMQ_REQ);

                                                    char address[32];
                                                    client_address_gen(m2.id, address);

                                                    //printf("[%d]: address=%s", my_id, address);

                                                    // printf("address=%s", address);
                                                    // printf("I am here\n");
                                                    // break;

                                                    check_neg_one(zmq_connect(requester,
address), "zmq_connect error");

                                                    send_msg(&m2, requester);
                                                recv_msg(&m2, requester);
                                                zmq_close(requester);

                                                wait(NULL);

                                                list_delete(childs, m.id);

                                                memcpy(&m2.cmd, "ok", sizeof(char) * 10);
                                                send_msg(&m2, responder);
                                               }
                                               else {
                                                void *requester = zmq_socket(context, ZMQ_REQ);

                                                int cur_id = echo(childs, m.id, requester);

            //printf("[%d]: cur_id = %d\n", my_id, cur_id);

            if (cur_id == -1 || ping(cur_id) == -1) {
               memcpy(&m.cmd, "bad", sizeof(char) * 10);
                                                send_msg(&m, responder);
```

```
            continue;
        }

        char address[32];
        client_address_gen(cur_id, address);
        check_neg_one(zmq_connect(requester, address), "zmq_connect error");
        send_msg(&m, requester);
        recv_msg(&m, requester);
        zmq_close(requester);

        memcpy(&m.cmd, "ok", sizeof(char) * 10);
                                        send_msg(&m, responder);
                                        }
    }
    else if (strcmp(m.cmd, "echo") == 0) {
                                        //printf("[%d-echo]: %s %d %d %s\n", my_id, m.cmd, m.id,
m.num, m.str);
                                        if ((list_find(childs, m.id) == 1 && ping(m.id) == 1) || my_id ==
m.id) {
                                         memcpy(&m.cmd, "ok", sizeof(char) * 10);
                                        send_msg(&m, responder);
                                        } else {
                                         void *requester = zmq_socket(context, ZMQ_REQ);
                                         int cur_id = echo(childs, m.id, requester);
                                         zmq_close(requester);

                                         //printf("[%d]264: cur_id = %d\n", my_id, cur_id);

                                         if (cur_id == -1 || ping(cur_id) == -1) {
                                                memcpy(&m.cmd, "bad", sizeof(char) * 10);
                                        send_msg(&m, responder);
                                        } else {
                                                memcpy(&m.cmd, "ok", sizeof(char) * 10);
                                        send_msg(&m, responder);
                                         }
                                        }
```

```
    } else if (strcmp(m.cmd, "heartbit") == 0) {
                                        int delay = m.num;
                                        void *requester = zmq_socket(context, ZMQ_REQ);
       for (int i = 0, sz = size(childs); i < sz; ++i) {
                                        int cur_id = list_get(childs, i);
                                        check_neg_one(cur_id, "get error");

                                        //printf("[%d]: delay=%d cur_id=%d\n", my_id, delay, cur_id);

                                        int answers = 0;
                                        for (int j = 0; j < 4; ++j) {
                                         if (ping(cur_id) == 1) {
                                                answers += 1;
                                         }
                                         usleep(delay * 1000);
                                        }

                                                        if (answers == 4) {

                                                                printf("[%d]: ready\n", cur_id);

                                                                Message m = {"heartbit", delay, 0, ""};

                                                                char address[32];
                                                                client_address_gen(cur_id, address);


                                        check_neg_one(zmq_connect(requester, address),
"zmq_connect error");

                                                                send_msg(&m, requester);
                                                                recv_msg(&m, requester);

                                                                zmq_disconnect(requester, address);
                                                        } else {
                                                                printf("[%d]: bad\n", cur_id);
```

```
                                        }
                                    }
                                    zmq_close(requester);

                                    memcpy(&m.cmd, "ok", sizeof(char) * 10);
            send_msg(&m, responder);
        }


                                }
                                zmq_close(responder);
    zmq_ctx_destroy(context);
    list_destroy(childs);

    printf("[%d] dying...\n", my_id);


                                return 0;
}
```

lab6_utils.c

```
#include "lab6_utils.h"
#include "list.h"

#include <zmq.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>


void send_msg(Message *m, void *socket)
{
    zmq_msg_t msg;
    zmq_msg_init(&msg);
    zmq_msg_init_size(&msg, sizeof(Message)); //check -1
```

```
    memcpy(zmq_msg_data(&msg), m, sizeof(Message)); //check NULL
    zmq_msg_send(&msg, socket, 0); //check -1
    zmq_msg_close(&msg);
}

void recv_msg(Message *m, void *socket)
{
    zmq_msg_t msg;
    zmq_msg_init(&msg);
    zmq_msg_recv(&msg, socket, 0); //check -1
    //m = (Message *)zmq_msg_data(&msg); // check NULL
    memcpy(m, zmq_msg_data(&msg), sizeof(Message));
    zmq_msg_close(&msg);
}

int *__z_function(char *str, int size)
{
    int *z = calloc(size, sizeof(int));
    //printf("%s", str);
    for (int i = 1, l = 0, r = 0; i < size; ++i) {
        if (i <= r) {
            z[i] = min(r - i + 1, z[i - l]);
        }
        while(i + z[i] < size && str[z[i]] == str[i + z[i]]) {
            ++z[i];
    //      printf("%d size=%d %c %c\n", i + z[i], size, str[z[i]], str[i + z[i]]);
        }
        if (i + z[i] - 1 > r) {
            l = i, r = i + z[i] - 1;
        }
    }
    return z;
}

void print_array(int size, int *arr)
{
    //size -= 1;
    for (int i = 0; i < size; ++i) {
```

```
        printf("%d ", arr[i]);
    }
    printf("\n");
}

void print_part_of_str(char *str, int from, int to, char *res, int *len)
{
    for (int i = from; i <= to; ++i) {
        *len += sprintf(res + *len, "%c", str[i]);
    }
}

void find_substrings(Message *m)
{
    int all_size = strlen(m->str);
    int *z       = __z_function(m->str, all_size);
    char *res = (char *) malloc(sizeof(char) * 1024);
    memset(res, 0, sizeof(char) * 1024);
    for (int i = m->num, len = 0; i < all_size; ++i) {
        if (z[i] == m->num) {
            len += sprintf(res + len, "(%d): ", i - m->num);
            print_part_of_str(m->str, m->num + 1, i - 1, res, &len);
            len += sprintf(res + len, "%s", GREEN);
            print_part_of_str(m->str, i, i + m->num - 1, res, &len);
            len += sprintf(res + len, "%s", END_COL);
            print_part_of_str(m->str, i + m->num, all_size - 1, res, &len);
        }
    }
    free(z);
    memcpy(m->str, res, sizeof(char) * 1024);
}

void client_address_gen(int id, char *address)
{
                                        memset(address, 0, 32);
                                        memcpy(address, ADDRESS_PATTERN,
sizeof(ADDRESS_PATTERN));
                                        char s[8];
```

```
                                memset(s, 0, 8);
                                sprintf(s, "%d", id);
                                strcat(address, s);
}

char *server_address_gen(int id)
{
                                char s[8];
                                char *c = malloc(sizeof(char) * 32);
                                strcat(c, "tcp://*:");
                                //memcpy(c, ADDRESS_PATTERN,
sizeof(ADDRESS_PATTERN));
                                //memset(s, 0, 8);
                                sprintf(s, "%d", id);
                                strcat(c, s);
                                return c;
}

int ping(int id)
{
                                char id_str[8];
                                memset(id_str, 0, 8);
                                sprintf(id_str, "%d", id);

                                char inproc_address[32];
                                memset(inproc_address, 0, 32);
                                memcpy(inproc_address, PING_PATTERN,
sizeof(PING_PATTERN));
                                strcat(inproc_address, id_str);

                                char address[32];
                                memset(address, 0, 32);
                                memcpy(address, ADDRESS_PATTERN,
sizeof(ADDRESS_PATTERN));
                                strcat(address, id_str);

                                //printf("[in-ping]: address=%s\n", address);
```

```
                                        void *context = zmq_ctx_new();
                                        void *requester = zmq_socket(context, ZMQ_REQ);
                                        zmq_connect(requester, address);

                                        zmq_socket_monitor(requester, inproc_address,
ZMQ_EVENT_CONNECTED | ZMQ_EVENT_CONNECT_RETRIED);
                                        void *pair_socket = zmq_socket(context, ZMQ_PAIR);
                                        zmq_connect(pair_socket, inproc_address);

                                        zmq_msg_t m;
                                        zmq_msg_init(&m);
                                        zmq_msg_recv(&m, pair_socket, 0);
                                        uint8_t* data = (uint8_t*)zmq_msg_data(&m);
   uint16_t event = *(uint16_t*)(data);

   zmq_close(requester);
   zmq_close(pair_socket);
   zmq_msg_close(&m);
   zmq_ctx_destroy(context);

   if (event == ZMQ_EVENT_CONNECT_RETRIED) {
                                        return -1;
   } else {
                                        return 1;
   }
}

int echo(List *childs, int id, void *socket) // вернет id узла, у которого в потомках есть искомый id
{
                                        for (int i = 0, sz = size(childs); i < sz; ++i) {
                                         int cur_id = list_get(childs, i);
                                         check_neg_one(cur_id, "echo error");

                                         //printf("[in-echo] cur=%d id=%d\n", cur_id, id);
                                         if (cur_id == id) {
                                                 //printf("[in-echo]: 158 returned\n");
                                                 return cur_id;
                                         }
```

```
                                        //Message m = {"echo", parent_id, id, ""};
                                        Message m = {"echo", 0, id, ""}; //id

                                        //printf("[echo]: %s %d %d\n", m.cmd, m.id, m.num);
                                        // return -1;

                                        char address[32];
                                        client_address_gen(cur_id, address);
                                        check_neg_one(zmq_connect(socket, address), "zmq_connect
error");
                                        send_msg(&m, socket);
                                        recv_msg(&m, socket);

                                        zmq_disconnect(socket, address);

                                        //printf("[in-echo]: %s %d %d\n", m.cmd, m.id, m.num);

                                        if (strcmp(m.cmd, "ok") == 0) {
                                                //printf("[in-echo]: 179 returned cur_id=%d\n", cur_id);
                                                return cur_id;
                                        }
                                        }
                                        return -1;
}
```

lab6_utils.h

```
#ifndef _LAB6_UTILS_H_
#define _LAB6_UTILS_H_

#include "list.h"

#define PORT 9000
//#define ADDRESS_PATTERN "tcp://localhost:"
#define ADDRESS_PATTERN "tcp://127.0.0.1:"
```

```
#define SERVER_PATTERN  "tcp://*:"
#define PING_PATTERN    "inproc://ping-"
#define MAXSIZE         1024
#define MIN_INDEXES     8
#define END_COL         "\x1b[0m"
#define GREEN           "\x1b[32m"
#define min(a, b) a > b ? b : a;

#define check_null(foo, msg) do { \
if (foo == NULL) { perror(msg); exit(EXIT_FAILURE); } \
} while(0);

#define check_neg_one(foo, msg) do { int __res = foo; \
if (__res == -1) { perror(msg); exit(EXIT_FAILURE); } \
} while(0);

typedef struct
{
   char cmd[10];
   int num;
   int id;
   char str[1024];
} Message;

void send_msg(Message *m, void *socket);
void recv_msg(Message *m, void *socket);

int *__z_function(char *str, int size);
void print_array(int size, int *arr);
void print_part_of_str(char *str, int from, int to, char *res, int *len);
void find_substrings(Message *m);

void client_address_gen(int id, char *address);
char *server_address_gen(int id);
int ping(int id);
int echo(List *childs, int id, void *socket);
```

```
#endif
```

list.c

```
#include "list.h"

#include <stdio.h>
#include <stdlib.h>

List *list_create()
{
                                        List *l = (List *) malloc(sizeof(List));
                                        l->values = (int *) malloc(sizeof(int));
                                        l->size = 0;
                                        return l;
}

int size(List *l)
{
                                        return l->size;
}

void __resize(List *l)
{
                                        l->values = realloc(l->values, sizeof(int) * (l->size));
}

void push_back(List *l, int num)
{
                                        l->size++;
                                        __resize(l);
                                        l->values[l->size - 1] = num;
}

void print_list(List *l)
{
                                        int sz = size(l);
                                        // if (sz == 0) {
```

```
//          return;
// }
for (int i = 0; i < sz; ++i) {
 printf("%d ", l->values[i]);
}
printf("\n");
}

void list_delete(List *l, int num)
{
int sz = size(l);
for (int i = 0; i < sz; ++i) {
 if (l->values[i] == num) {
        for (int j = i; j < sz - 1; ++j) {
                l->values[j] = l->values[j + 1];
        }
        l->size--;
        break;
 }
}
if (l->size != 0) {
 __resize(l);
}
}

int list_find(List *l, int num)
{
int sz = size(l);
for (int i = 0; i < sz; ++i) {
 if (l->values[i] == num) {
        return 1;
 }
}
return -1;
}


void list_destroy(List *l)
```

```
{
                                        free(l->values);
                                        free(l);
}

int list_get(List *l, int ind)
{
                                        int sz = size(l);
                                        if (ind < sz) {
                                         return l->values[ind];
                                        } else {
                                         return -1;
                                        }
}
```

list.h

```
#ifndef _LIST_H_
#define _LIST_H_

typedef struct
{
                                        int size;
                                        int *values;
} List;

List *list_create();
int size(List *l);
void _resize(List *l);
void push_back(List *l, int num);
void print_list(List *l);
void list_delete(List *l, int num);
int list_find(List *l, int num);
void list_destroy(List *l);
int list_get(List *l, int ind);

#endif
```

Демонстрация работы программы

```
user@brokiloene:~/Desktop/all/os/lab_6/build$ ./manage_node
> create 1 -1
> computing_node № 9001 with pid [14116] has been created
> create 2 1
ok: [9002] was created by [9001]
> create 3 2
ok: [9003] was created by [9002] via [9001]
> exec 3
a
abadaba
ok:
(1): abadaba
(3): abadaba
(5): abadaba
(7): abadaba

produced by [9003]
remove 3
[9003] dying...
> ok: [9003] with pid 14133 has been removed via [9001]
> remove 20
bad id
> heartbit 2
[9001]: ready
[9002]: ready
```

Выводы

В ходе выполнения лабораторной работы изучил основы работы с очередями сообщений ZMQ и реализовал программу с использованием этой библиотеки. В процессе открыл для себя много нового. Почти вся информация бралась из официальной документации.