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: Уваров А.П.

Группа: М8О–206Б–19

Вариант: 31

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

 Управлении серверами сообщений (№6)

 Применение отложенных вычислений (№7)

 Интеграция программных систем друг с другом (№8)

**Задание**

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 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.

Вариант 31: pingall, клиенты хранятся в бинарном дереве, функции времени(start/stop/time)

**Общие сведения о программе**

Для работы с очередями используется ZMQ, программа собирается при помощи Makefile. Управляющий узел – server, вычислительные узлы – client. В программе используются следующие системные вызовы:

1. **kill** – убивает процесс с pid – первый аргумент и посылает сигнал – второй аргумент.
2. **socket.setsockopt** – устанавливает настройки для сокета.
3. **zmq::context\_t –** создает ZMQ контекст(хранилище сокетов).
4. **zmq::socket\_t –** создает ZMQ сокет.
5. **zmq::message\_t** – создает ZMQ сообщение.
6. **socket.send** – отправляет ZMQ сообщение на socket.
7. **bind –** подключение socket на запись.
8. **connect –** подключение socket на чтение.
9. **execl –** выполняет указанный файл c аргументами, которые равны аргументам в указанном файле, аргументы пишутся до null.
10. **fork** – создает новый процесс.

**Общий метод и алгоритм решения**.

Создаем по сервер – исполняющий узел, дальше делаем fork, в дочернем процессе при помощи execv запускаем client, а с родителя с сервера отсылаем сообщение, внутри клиента также создаются сокеты – левый и правый и на них отправляются сообщения с родителя, а родитель получает сообщения от детей и так по всему дереву. Исполняющий узел получает сообщение выполняет команду и отправляет ответ вверх по дереву до управляющего узла.

**Основные файлы программы**

**Makefile:**

**CC = gcc**

**CC\_FLAGS = -pedantic -Wall**

**LD\_FLAGS = -lzmq -L.**

**CC\_LIB\_FLAGS = -c -pedantic -Wall -fPIC**

**LD\_LIB\_FLAGS = -shared -lzmq**

**all: server client**

**server : server.c libtools.so**

**$(CC) $(CC\_FLAGS) server.c -o server $(LD\_FLAGS) -ltools**

**client : client.c libtools.so**

**$(CC) $(CC\_FLAGS) client.c -o client $(LD\_FLAGS) -ltools**

**libtools.so : zmq\_comm.o**

**$(CC) zmq\_comm.o -o libtools.so $(LD\_LIB\_FLAGS)**

**zmq\_comm.o : zmq\_comm.c**

**$(CC) $(CC\_LIB\_FLAGS) zmq\_comm.c**

**clean:**

**rm -f \*.o \*.so server client**

**server.c:**

**#include <stdio.h>**

**#include <string.h>**

**#include <signal.h>**

**#include <stdbool.h>**

**#include "zmq\_comm.h"**

**static void \* CONTEXT;**

**static void \* PUBLISHER;**

**static void \* SUBSCRIBER;**

**char PUB\_ENDPOINT[MAX\_LEN];**

**char SUB\_ENDPOINT[MAX\_LEN];**

**bool FIRST\_CLIENT\_CONNECTED;**

**int FIRST\_CLIENT\_ID;**

**pid\_t SERVER\_PID;**

**int FEEDBACK\_TIME;**

**void init\_server() {**

**FEEDBACK\_TIME = 1000;**

**CONTEXT = create\_zmq\_context();**

**PUBLISHER = create\_zmq\_socket(CONTEXT, ZMQ\_PUB);**

**create\_endpoint(PUB\_ENDPOINT, 0, server\_pub); // PUB\_ENDP - путь на котором создастся сокет**

**bind\_zmq\_socket(PUBLISHER, PUB\_ENDPOINT); // Сокет должен писать в путь PUB\_ENDPOINT**

**SUBSCRIBER = create\_zmq\_socket(CONTEXT, ZMQ\_SUB);**

**zmq\_setsockopt(SUBSCRIBER, ZMQ\_RCVTIMEO, &FEEDBACK\_TIME, sizeof(FEEDBACK\_TIME));**

**FIRST\_CLIENT\_CONNECTED = false;**

**}**

**void deinit\_server() {**

**unbind\_zmq\_socket(PUBLISHER, PUB\_ENDPOINT);**

**close\_zmq\_socket(PUBLISHER);**

**if (FIRST\_CLIENT\_CONNECTED) {**

**disconnect\_zmq\_socket(SUBSCRIBER, SUB\_ENDPOINT);**

**}**

**close\_zmq\_socket(SUBSCRIBER);**

**destroy\_zmq\_context(CONTEXT);**

**}**

**void term\_clients() {**

**if (FIRST\_CLIENT\_CONNECTED) {**

**message msg;**

**msg.cmd = kill\_all;**

**send\_msg(PUBLISHER, &msg);**

**}**

**}**

**void send\_exec\_msg( const int to\_id, const cmd\_type type) {**

**message msg;**

**create\_msg(&msg, type, to\_id, 0);**

**send\_msg(PUBLISHER, &msg);**

**}**

**void send\_create\_msg(const int to\_create\_id) {**

**message msg;**

**create\_msg(&msg, create, to\_create\_id, 0);**

**send\_msg(PUBLISHER, &msg);**

**}**

**void send\_ping\_msg(const int to\_id) {**

**message msg;**

**create\_msg(&msg, ping, to\_id, 0);**

**send\_msg(PUBLISHER, &msg);**

**}**

**void send\_remove\_msg(const int to\_id) {**

**message msg;**

**create\_msg(&msg, delete, to\_id, 0);**

**send\_msg(PUBLISHER, &msg);**

**}**

**int ping\_client(const int client\_id) {**

**send\_ping\_msg(client\_id);**

**message msg;**

**if (!get\_msg(SUBSCRIBER, &msg)) {**

**return 0;**

**}**

**if (msg.to\_id == -1 && msg.cmd == ping) {**

**return msg.value;**

**}**

**fprintf(stderr, "[%d] SERVER ERROR: received another msg in ping\_client\n", SERVER\_PID); fflush(stdout);**

**print\_message(&msg);**

**return 0;**

**}**

**void create\_child\_process(const int client\_id) {**

**if (client\_id < 1) {**

**fprintf(stderr, "[%d] SERVER ERROR: invalid id\n", SERVER\_PID);**

**return;**

**}**

**if (!FIRST\_CLIENT\_CONNECTED) {**

**FIRST\_CLIENT\_ID = client\_id;**

**char parent\_sub\_endpoint[MAX\_LEN]; // Подписка клиента на родителя**

**char parent\_pub\_endpoint[MAX\_LEN]; // Паблиш клиента родителю**

**create\_endpoint(parent\_sub\_endpoint, 0, server\_pub);**

**create\_endpoint(parent\_pub\_endpoint, client\_id, client\_parent\_pub);**

**int fork\_pid = fork();**

**if (fork\_pid < 0) {**

**fprintf(stderr, "[%d] SERVER ERROR: Unable to fork a child\n", SERVER\_PID);**

**exit(10);**

**} else if (fork\_pid == 0) {**

**char client\_id\_str[MAX\_LEN];**

**sprintf(client\_id\_str, "%d", client\_id);**

**// argv = [id, parent\_id, parent\_sub\_end, parent\_pub\_end]**

**execl(CLIENT, CLIENT, client\_id\_str, "-1", parent\_sub\_endpoint, parent\_pub\_endpoint, NULL);**

**} else {**

**strcpy(SUB\_ENDPOINT, parent\_pub\_endpoint);**

**connect\_zmq\_socket(SUBSCRIBER, SUB\_ENDPOINT);**

**FIRST\_CLIENT\_CONNECTED = true;**

**printf("[%d] OK: %d\n", SERVER\_PID, fork\_pid);**

**}**

**} else {**

**int client\_pid = ping\_client(client\_id);**

**if (client\_pid) {**

**fprintf(stderr, "[%d] Error: already exist with pid = \"%d\"\n", SERVER\_PID, client\_pid);**

**} else {**

**send\_create\_msg(client\_id);**

**int client\_pid = ping\_client(client\_id);**

**if (!client\_pid) {**

**fprintf(stderr, "[%d] Error: client wasn't created\n", SERVER\_PID);**

**} else {**

**printf("[%d] OK: %d\n", SERVER\_PID, client\_pid);**

**}**

**}**

**}**

**}**

**void exec\_command(const int client\_id, const cmd\_type cmd) {**

**if (!ping\_client(client\_id)) {**

**fprintf(stderr, "[%d] Error: not found\n", SERVER\_PID);**

**return;**

**}**

**send\_exec\_msg(client\_id, cmd);**

**message msg;**

**if (!get\_msg(SUBSCRIBER, &msg)) {**

**fprintf(stderr, "[%d] Error: client haven't responed\n", SERVER\_PID);**

**return;**

**}**

**if (msg.to\_id == -1 && msg.cmd == cmd) {**

**if (cmd == timer\_display) {**

**printf("[%d] OK:%d: %d\n", SERVER\_PID, client\_id, msg.value);**

**} else {**

**printf("[%d] OK:%d\n", SERVER\_PID, client\_id);**

**}**

**return;**

**}**

**fprintf(stderr, "[%d] SERVER ERROR: received another msg in exec\_command\n", SERVER\_PID); fflush(stdout);**

**print\_message(&msg);**

**return;**

**}**

**void remove\_client(const int client\_id) {**

**int ping\_res = ping\_client(client\_id);**

**if (!ping\_res) {**

**fprintf(stderr, "[%d] Error: not found\n", SERVER\_PID);**

**return;**

**}**

**send\_remove\_msg(client\_id);**

**if (FIRST\_CLIENT\_ID == client\_id) {**

**disconnect\_zmq\_socket(SUBSCRIBER, SUB\_ENDPOINT);**

**memset(SUB\_ENDPOINT, 0, MAX\_LEN);**

**FIRST\_CLIENT\_CONNECTED = false;**

**}**

**if (!ping\_client(client\_id)) {**

**printf("[%d] OK\n", SERVER\_PID);**

**} else {**

**fprintf(stderr, "[%d] Error: client wasn't removed\n", SERVER\_PID);**

**}**

**}**

**void skip\_line() {**

**char cc = getchar();**

**while (cc != '\n' && cc != EOF) { cc = getchar(); }**

**}**

**void input\_loop() {**

**char cmd[MAX\_LEN];**

**while(scanf("%s", cmd) != EOF){**

**int id;**

**if(scanf("%d",&id) == EOF){**

**printf("[%d] Shutting down server...\n", SERVER\_PID);**

**break;**

**}**

**if (!strcmp(cmd, "create")) {**

**create\_child\_process(id);**

**} else**

**if (!strcmp(cmd, "remove")) {**

**remove\_client(id);**

**} else**

**if (!strcmp(cmd, "exec")) {**

**char exec\_cmd[MAX\_LEN];**

**int scanf\_res = scanf("%s", exec\_cmd);**

**if (scanf\_res == EOF) {**

**printf("[%d] Shutting down server.\n", SERVER\_PID);**

**break;**

**} else if (scanf\_res == 0) {**

**printf("[%d] Invalid exec command\n", SERVER\_PID);**

**skip\_line();**

**continue;**

**}**

**cmd\_type type;**

**if (!strcmp(exec\_cmd, "start")) {**

**type = timer\_start;**

**} else**

**if (!strcmp(exec\_cmd, "stop")) {**

**type = timer\_stop;**

**} else**

**if (!strcmp(exec\_cmd, "time")) {**

**type = timer\_display;**

**} else {**

**printf("[%d] Invalid exec command\n", SERVER\_PID);**

**continue;**

**}**

**exec\_command(id, type);**

**} else**

**if (!strcmp(cmd, "ping")) {**

**if (!ping\_client(id)) {**

**fprintf(stderr, "[%d] Error: not found\n", SERVER\_PID);**

**} else {**

**printf("[%d] OK: 1\n", SERVER\_PID);**

**}**

**} else**

**if (!strcmp(cmd, "test")) {**

**if (id == 0) {**

**printf("[%d] I'm server\n"**

**"\t publisher = \"%s\"\n"**

**"\t subscriber = \"%s\"\n",**

**SERVER\_PID, PUB\_ENDPOINT, SUB\_ENDPOINT);**

**} else {**

**message msg;**

**create\_msg(&msg, test, id, 0);**

**send\_msg(PUBLISHER, &msg);**

**}**

**} else {**

**printf("[%d] Invalid command\n", SERVER\_PID);**

**skip\_line();**

**continue;**

**}**

**}**

**}**

**int main (int argc, char const \*argv[]) {**

**SERVER\_PID = getpid();**

**printf("[%d] Starting server...\n", SERVER\_PID);**

**init\_server();**

**input\_loop();**

**term\_clients();**

**deinit\_server();**

**return 0;**

**}**

**client.c:**

**#include <stdio.h>**

**#include <signal.h>**

**#include <string.h>**

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

**#include "zmq\_comm.h"**

**// context and sockets**

**static void \* CONTEXT;**

**static void \* SUBSCRIBER;**

**static void \* PARENT\_PUB;**

**static void \* RIGHT\_PUB;**

**static void \* LEFT\_PUB;**

**// SOCKETS ENDPOINT**

**// for subscriber socket**

**static char PARENT\_SUB\_ENDPOINT [MAX\_LEN]; // recieve by parent**

**static char LEFT\_SUB\_ENDPOINT [MAX\_LEN];**

**static char RIGHT\_SUB\_ENDPOINT [MAX\_LEN];**

**static char PARENT\_PUB\_ENDPOINT [MAX\_LEN]; // recieve by parent**

**static char LEFT\_PUB\_ENDPOINT [MAX\_LEN];**

**static char RIGHT\_PUB\_ENDPOINT [MAX\_LEN];**

**static int PARENT\_ID;**

**static int CLIENT\_PID;**

**static int CLIENT\_ID;**

**bool HAS\_LEFT;**

**bool HAS\_RIGHT;**

**struct timeval START, END;**

**void init\_client() {**

**create\_endpoint(LEFT\_PUB\_ENDPOINT, CLIENT\_ID, client\_left\_pub);**

**create\_endpoint(RIGHT\_PUB\_ENDPOINT, CLIENT\_ID, client\_right\_pub);**

**CONTEXT = create\_zmq\_context();**

**SUBSCRIBER = create\_zmq\_socket(CONTEXT, ZMQ\_SUB);**

**connect\_zmq\_socket(SUBSCRIBER, PARENT\_SUB\_ENDPOINT);**

**PARENT\_PUB = create\_zmq\_socket(CONTEXT, ZMQ\_PUB);**

**bind\_zmq\_socket(PARENT\_PUB, PARENT\_PUB\_ENDPOINT);**

**RIGHT\_PUB = create\_zmq\_socket(CONTEXT, ZMQ\_PUB);**

**bind\_zmq\_socket(RIGHT\_PUB, RIGHT\_PUB\_ENDPOINT);**

**LEFT\_PUB = create\_zmq\_socket(CONTEXT, ZMQ\_PUB);**

**bind\_zmq\_socket(LEFT\_PUB, LEFT\_PUB\_ENDPOINT);**

**}**

**void deinit\_client() {**

**disconnect\_zmq\_socket(SUBSCRIBER, PARENT\_SUB\_ENDPOINT);**

**if (HAS\_LEFT) {**

**disconnect\_zmq\_socket(SUBSCRIBER, LEFT\_SUB\_ENDPOINT);**

**}**

**if (HAS\_RIGHT) {**

**disconnect\_zmq\_socket(SUBSCRIBER, RIGHT\_SUB\_ENDPOINT);**

**}**

**close\_zmq\_socket(SUBSCRIBER);**

**unbind\_zmq\_socket(PARENT\_PUB, PARENT\_PUB\_ENDPOINT);**

**unbind\_zmq\_socket(LEFT\_PUB, LEFT\_PUB\_ENDPOINT);**

**unbind\_zmq\_socket(RIGHT\_PUB, RIGHT\_PUB\_ENDPOINT);**

**close\_zmq\_socket(PARENT\_PUB);**

**close\_zmq\_socket(LEFT\_PUB);**

**close\_zmq\_socket(RIGHT\_PUB);**

**}**

**void create\_child\_process(const int child\_id) {**

**char child\_sub\_endpoint[MAX\_LEN]; // parent --> child**

**bool is\_left = child\_id < CLIENT\_ID;**

**if (is\_left) {**

**strcpy(child\_sub\_endpoint, LEFT\_PUB\_ENDPOINT);**

**} else {**

**strcpy(child\_sub\_endpoint, RIGHT\_PUB\_ENDPOINT);**

**}**

**char child\_pub\_endpoint[MAX\_LEN]; // parent <-- child**

**create\_endpoint(child\_pub\_endpoint, child\_id, client\_parent\_pub);**

**int fork\_pid = fork();**

**if (fork\_pid < 0) {**

**fprintf(stderr, "[%d] Client error: Unable to fork a child\n", CLIENT\_PID);**

**exit(10);**

**} else if (fork\_pid == 0) {**

**char child\_id\_str[MAX\_LEN];**

**char parent\_id\_str[MAX\_LEN];**

**sprintf(child\_id\_str, "%d", child\_id);**

**sprintf(parent\_id\_str, "%d", CLIENT\_ID);**

**// argv = [id, parent\_id, parent\_sub\_end, parent\_pub\_end]**

**execl(CLIENT, CLIENT, child\_id\_str, parent\_id\_str, child\_sub\_endpoint, child\_pub\_endpoint, NULL);**

**} else {**

**if (is\_left) {**

**strcpy(LEFT\_SUB\_ENDPOINT, child\_pub\_endpoint); // Cur Client reads from left child**

**HAS\_LEFT = true;**

**} else {**

**strcpy(RIGHT\_SUB\_ENDPOINT, child\_pub\_endpoint);**

**HAS\_RIGHT = true;**

**}**

**bind\_zmq\_socket(SUBSCRIBER, child\_pub\_endpoint);**

**}**

**}**

**void change\_subscription(const int from\_id) {**

**if (from\_id < CLIENT\_ID) {**

**HAS\_LEFT = false;**

**disconnect\_zmq\_socket(SUBSCRIBER, LEFT\_SUB\_ENDPOINT);**

**memset(LEFT\_SUB\_ENDPOINT, 0, MAX\_LEN);**

**} else {**

**HAS\_RIGHT = false;**

**disconnect\_zmq\_socket(SUBSCRIBER, RIGHT\_SUB\_ENDPOINT);**

**memset(RIGHT\_SUB\_ENDPOINT, 0, MAX\_LEN);**

**}**

**}**

**void client\_loop() {**

**for (;;) {**

**message msg;**

**get\_msg(SUBSCRIBER, &msg);**

**if (msg.to\_id == -1) {**

**send\_msg(PARENT\_PUB, &msg);**

**continue;**

**}**

**if (msg.cmd != create && msg.cmd != kill\_all) {**

**if (msg.to\_id != CLIENT\_ID) {**

**if (msg.to\_id < CLIENT\_ID) {**

**send\_msg(LEFT\_PUB, &msg);**

**} else {**

**send\_msg(RIGHT\_PUB, &msg);**

**}**

**continue;**

**}**

**}**

**// parsing msg**

**if (msg.cmd == create) {**

**if (msg.to\_id < CLIENT\_ID && HAS\_LEFT) {**

**send\_msg(LEFT\_PUB, &msg);**

**} else**

**if (msg.to\_id > CLIENT\_ID && HAS\_RIGHT) {**

**send\_msg(RIGHT\_PUB, &msg);**

**} else {**

**create\_child\_process(msg.to\_id);**

**}**

**} else if (msg.cmd == delete) {**

**if (PARENT\_ID != -1) {**

**msg.to\_id = PARENT\_ID;**

**msg.value = CLIENT\_ID;**

**msg.cmd = change\_sub;**

**send\_msg(PARENT\_PUB, &msg);**

**}**

**msg.cmd = kill\_all;**

**send\_msg(LEFT\_PUB, &msg);**

**send\_msg(RIGHT\_PUB, &msg);**

**raise(SIGTERM);**

**} else if (msg.cmd == kill\_all) {**

**send\_msg(LEFT\_PUB, &msg);**

**send\_msg(RIGHT\_PUB, &msg);**

**raise(SIGTERM);**

**} else if (msg.cmd == ping) {**

**msg.to\_id = -1;**

**msg.value = CLIENT\_PID;**

**send\_msg(PARENT\_PUB, &msg);**

**} else if (msg.cmd == change\_sub) {**

**change\_subscription(msg.value);**

**} else if (msg.cmd == timer\_start) {**

**gettimeofday(&START, NULL);**

**msg.to\_id = -1;**

**send\_msg(PARENT\_PUB, &msg);**

**} else if (msg.cmd == timer\_stop) {**

**gettimeofday(&END, NULL);**

**msg.to\_id = -1;**

**send\_msg(PARENT\_PUB, &msg);**

**} else if (msg.cmd == timer\_display) {**

**int sec = END.tv\_sec - START.tv\_sec;**

**int microsec = END.tv\_usec - START.tv\_usec;**

**if (microsec < 0) {**

**--sec;**

**microsec += 1000000;**

**}**

**int elapsed = sec\*1000 + microsec/1000;**

**msg.to\_id = -1;**

**msg.value = elapsed;**

**send\_msg(PARENT\_PUB, &msg);**

**} else {**

**printf("[%d] Client error: unknown command\n", CLIENT\_PID);**

**print\_message(&msg);**

**continue;**

**}**

**}**

**}**

**// [id, parent\_id, parent\_sub\_end, parent\_pub\_end]**

**int main(int argc, char const \*argv[]) {**

**CLIENT\_PID = getpid();**

**CLIENT\_ID = atoi(argv[1]);**

**PARENT\_ID = atoi(argv[2]);**

**strcpy(PARENT\_SUB\_ENDPOINT, argv[3]);**

**strcpy(PARENT\_PUB\_ENDPOINT, argv[4]);**

**init\_client();**

**client\_loop();**

**deinit\_client();**

**return 0;**

**}**

**Zmq\_comm.c:**

**#include "zmq\_comm.h"**

**void message\_cpy(message\* from, message\* to) {**

**to->cmd = from->cmd;**

**to->to\_id = from->to\_id;**

**to->value = from->value;**

**}**

**void create\_msg(message\* msg, const cmd\_type cmd, const int to\_id, const int value) {**

**msg->cmd = cmd;**

**msg->to\_id = to\_id;**

**msg->value = value;**

**}**

**void print\_message(message\* msg) {**

**printf("\tmsg content:\n"**

**"\t cmd = \"%d\"\n"**

**"\tto\_id = \"%d\"\n"**

**"\tvalue = \"%d\"\n",**

**msg->cmd, msg->to\_id, msg->value);**

**fflush(stdout);**

**}**

**void send\_msg(void\* socket, const message\* msg) {**

**zmq\_msg\_t zmq\_msg;**

**zmq\_msg\_init(&zmq\_msg);**

**zmq\_msg\_init\_size(&zmq\_msg, sizeof(\*msg));**

**memcpy(zmq\_msg\_data(&zmq\_msg), msg, sizeof(\*msg));**

**if (!zmq\_msg\_send(&zmq\_msg, socket, 0)) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR send\_msg \n");**

**exit(ERR\_ZMQ\_SEND);**

**}**

**zmq\_msg\_close(&zmq\_msg);**

**}**

**// 1 on success**

**int get\_msg(void\* socket, message\* msg) {**

**zmq\_msg\_t zmq\_msg;**

**zmq\_msg\_init(&zmq\_msg);**

**zmq\_msg\_init\_size(&zmq\_msg, sizeof(message));**

**int rc = zmq\_msg\_recv(&zmq\_msg, socket, 0);**

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

**return 0;**

**}**

**memcpy(msg, zmq\_msg\_data(&zmq\_msg), sizeof(message));**

**zmq\_msg\_close(&zmq\_msg);**

**return 1;**

**}**

**void create\_endpoint(char\* endpoint, int id, endpoint\_type type) {**

**char id\_str[MAX\_LEN];**

**sprintf(id\_str, "%d", id);**

**char\* pattern;**

**if (type == server\_pub) {**

**strcpy(endpoint, SERVER\_SOCKET\_PUB);**

**return;**

**} else if (type == client\_left\_pub) {**

**pattern = CLIENT\_LEFT\_PUB\_PATTERN;**

**} else if (type == client\_right\_pub) {**

**pattern = CLIENT\_RIGHT\_PUB\_PATTERN;**

**} else if (type == client\_parent\_pub) {**

**pattern = CLIENT\_PARENT\_PUB\_PATTERN;**

**}**

**strcpy(endpoint, pattern);**

**strcat(endpoint, id\_str);**

**}**

**///**

**void\* create\_zmq\_context() {**

**void\* context = zmq\_ctx\_new();**

**if (context == NULL) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_ctx\_new ");**

**exit(ERR\_ZMQ\_CTX);**

**}**

**return context;**

**}**

**void disconnect\_zmq\_socket(void\* socket, char\* endpoint) {**

**if (zmq\_disconnect(socket, endpoint) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_disconnect ");**

**exit(ERR\_ZMQ\_DISCONNECT);**

**}**

**}**

**void connect\_zmq\_socket(void\* socket, char\* endpoint) {**

**if (zmq\_connect(socket, endpoint) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_connect ");**

**exit(ERR\_ZMQ\_CONNECT);**

**}**

**}**

**void bind\_zmq\_socket(void\* socket, char\* endpoint) {**

**if (zmq\_bind(socket, endpoint) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_bind ");**

**exit(ERR\_ZMQ\_BIND);**

**}**

**}**

**void unbind\_zmq\_socket(void\* socket, char\* endpoint) {**

**if (zmq\_unbind(socket, endpoint) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_unbind ");**

**exit(ERR\_ZMQ\_BIND);**

**}**

**}**

**void\* create\_zmq\_socket(void\* context, const int type) {**

**void\* socket = zmq\_socket(context, type);**

**if (socket == NULL) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_socket ");**

**exit(ERR\_ZMQ\_SOCKET);**

**}**

**if (type == ZMQ\_SUB) {**

**zmq\_setsockopt(socket, ZMQ\_SUBSCRIBE, 0, 0);**

**}**

**return socket;**

**}**

**void reconnect\_zmq\_socket(void\* socket, char\* from, char\* to) {**

**disconnect\_zmq\_socket(socket, from);**

**connect\_zmq\_socket(socket, to);**

**}**

**void close\_zmq\_socket(void\* socket) {**

**if (zmq\_close(socket) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_close ");**

**exit(ERR\_ZMQ\_CLOSE);**

**}**

**}**

**void destroy\_zmq\_context(void\* context) {**

**if (zmq\_ctx\_destroy(context) != 0) {**

**fprintf(stderr, "[%d] ", getpid());**

**perror("ERROR zmq\_ctx\_destroy ");**

**exit(ERR\_ZMQ\_CLOSE);**

**}**

**}**

**zmq\_comm.h:**

**#pragma once**

**#include <stdbool.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <stdlib.h>**

**#include <errno.h>**

**#include <zmq.h>**

**#define MAX\_LEN 64**

**#define ERR\_ZMQ\_CTX 100**

**#define ERR\_ZMQ\_SOCKET 101**

**#define ERR\_ZMQ\_BIND 102**

**#define ERR\_ZMQ\_CLOSE 102**

**#define ERR\_ZMQ\_CTX\_DESTROY 103**

**#define ERR\_ZMQ\_CONNECT 104**

**#define ERR\_ZMQ\_DISCONNECT 105**

**#define ERR\_ZMQ\_SEND 106**

**#define SERVER\_SOCKET\_PUB "ipc:///tmp/server\_pub"**

**#define CLIENT\_PARENT\_PUB\_PATTERN "ipc:///tmp/client\_parent\_pub\_"**

**#define CLIENT\_LEFT\_PUB\_PATTERN "ipc:///tmp/client\_left\_pub\_"**

**#define CLIENT\_RIGHT\_PUB\_PATTERN "ipc:///tmp/client\_right\_pub\_"**

**#define CLIENT "./client"**

**#define CLIENT\_SIG\_ERR 3**

**#define SERVER\_SIG\_ERR 4**

**typedef enum {**

**create, // to\_id = id to create**

**delete,**

**kill\_all, // no to\_id**

**timer\_start,**

**timer\_stop,**

**timer\_display,**

**ping,**

**change\_sub,**

**test**

**} cmd\_type;**

**typedef struct {**

**cmd\_type cmd;**

**int to\_id;**

**int value;**

**} message;**

**typedef enum {**

**server\_pub,**

**client\_left\_pub,**

**client\_right\_pub,**

**client\_parent\_pub**

**} endpoint\_type;**

**void message\_cpy(message\* from, message\* to);**

**void create\_msg(message\* msg, const cmd\_type cmd, const int to\_id, const int value);**

**void print\_message(message\* msg);**

**void create\_zmq\_msg(zmq\_msg\_t\* zmq\_msg, const message\* msg);**

**void send\_msg(void\* socket, const message\* msg);**

**int get\_msg(void\* socket, message\* msg);**

**void create\_endpoint(char\* endpoint, int id, endpoint\_type type);**

**void\* create\_zmq\_context();**

**void connect\_zmq\_socket(void\* socket, char\* endpoint);**

**void disconnect\_zmq\_socket(void\* socket, char\* endpoint);**

**void bind\_zmq\_socket(void\* socket, char\* endpoint);**

**void unbind\_zmq\_socket(void\* socket, char\* endpoint);**

**void reconnect\_zmq\_socket(void\* socket, char\* from, char\* to);**

**void\* create\_zmq\_socket(void\* context, const int type);**

**void close\_zmq\_socket(void\* socket);**

**void destroy\_zmq\_context(void\* context);**

**Пример работы**

**andrey@andrey-VivoBook:~ /Рабочий стол/OC/OChelp/6/os\_lab\_678/src$ ./server**

**[21153] Starting server...**

**create 1**

**[21153] OK: 21158**

**remove 1**

**[21153] OK**

**exec 1 start**

**[21153] Error: not found**

**create 1**

**[21153] OK: 21161**

**exec 1 start**

**[21153] OK:1**

**exec 1 stop**

**[21153] OK:1**

**exec 1 time**

**[21153] OK:1: 3114**

**remove 1**

**[21153] OK**

**exec 1 start**

**[21153] Error: not found**

**remove 1**

**[21153] Error: not found**

**Вывод**

Я познакомился с сокетами, с передачами сообщений, с такой командой как execl, узнал про ipc, tcp, применял только icp, но все же узнал про tcp и zmq. Узнал, что есть немного другая реализация сокетов, и она мне понравилась больше, чем zmq. В zmq, на мой взгляд, слишком много трудностей. Реализовал клиентов в структуре бинарного дерева, совместил сокеты с процессами(fork). Узнал о разновидностях exec.