Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Операционные системы»

Лабораторная работа №6

Тема: Очереди сообщений

Студент: Глушатов И.С. Группа:
М8О-207Б-19 Преподаватель:
Миронов Е. С. Дата:
Оценка:

Москва, 2020

Цель работы: Целью является приобретение практических навыков в управлении серверами сообщений, применение отложенных вычислений и интеграция программных систем друг с другом.

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

Список основных поддерживаемых команд:

1. Создание нового вычислительного узла (create id)
2. Удаление существующего вычислительного узла (remove id)
3. Исполнение команды на вычислительном узле (exec id key [value])
4. Проверка доступности узлов (heartbit/ping)

Вариант 3-2-3:

1. Топология – бинарное дерево поиска
2. Тип команды – добавление и удаление из локального словаря
3. Тип проверки доступности узлов – heartbit + ping

Сервер в бесконечном цикле принимает команды пользователя, что является по факту его единственной задачей, хотя помимо этого сервер хранит словарь для команды heartbit, где содержится время последнего сообщения от i’того узла. Все ответы узлов о выполненной (удачно или нет) работе отправляются серверу в отдельный поток-обработчик.
Клиенты-узлы же принимают сообщение и решают, что с ним делать дальше. В зависимости от того, кому это сообщение принадлежит, оно либо остается и

обрабатывается, либо отправляется дальше вниз по абстрактному дереву из топологии. Для команды heartbit в узлах тоже выделяется отдельный поток, который отправляет сообщение о том, что узел жив и спит заданное количество времени, ожидая следующую отправку.

Листинг программы

server.cpp

```
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "zmq.h"
#include <unistd.h>
#include <pthread.h>
#include <string>
#include <map>

typedef struct MD {
char message[128];
int clientId;
char name[128] {'\0'};
int value = -1;
} MessageData;

const char* first_client = "tcp://127.0.0.1:4040";

void ZMQ_SEND(MessageData *md, void* socket_request) {
zmq_msg_t zmqMessage;
zmq_msg_init_size(&zmqMessage, sizeof(MessageData));
memcpy(zmq_msg_data(&zmqMessage), md, sizeof(MessageData));
int send = zmq_msg_send(&zmqMessage, socket_request, ZMQ_DONTWAIT);
if (send == -1) perror("Zmq_msg_send");
zmq_msg_close(&zmqMessage);
}
```

```
bool start = true;
std::map<int, unsigned int> times;

void* handler(void* socket_answers) {

while (start) {
	zmq_msg_t message;
	zmq_msg_init(&message);
	int recv = zmq_msg_recv(&message, socket_answers, ZMQ_DONTWAIT);
	if (recv != -1) {
		MessageData *md = (MessageData *)zmq_msg_data(&message);
		if (strcmp(md->message, "heartbit") == 0) {
			times.insert_or_assign(md->clientId, time(NULL));
		} else {
			if (strcmp(md->name, "") == 0) {
				if (md->clientId != -1) {
					printf("%s%d\n", md->message, md->clientId);
				} else {
					printf("%s\n", md->message);
				}
			} else {
				printf("%s%d: '%s' not found\n", md->message, md->clientId, md-
>name);
			}
		}
	}
	zmq_msg_close(&message);
}

return NULL;
}

int main(int argc, char const *argv[])
{
void* context_answers = zmq_ctx_new(); // контекст для ответов
void* context_request = NULL; // контекст для отправки запросов пользователя
if (context_answers == NULL) perror("zmq_ctx_new");

printf("Producer starting...\n");
```

```
void* socket_answers = zmq_socket(context_answers, ZMQ_PULL); // инициализация сокета для ответов
void* socket_request = NULL; // сокет для отправки запросов
if (socket_answers == NULL) perror("zmq_socket");

int bind_sa = zmq_bind(socket_answers, "tcp://127.0.0.1:4004"); // создание сокета для ответов
if (bind_sa == -1) perror("zmq_bind");

pthread_t hand;
if (pthread_create(&hand, NULL, handler, socket_answers) != 0) perror("Поток не смог создаться");

int child = -1;
int last_heartbit_time = -1;

for (;;)
{
	MessageData md;
	int status;

	status = scanf("%s", md.message);
	if (status <= 0) {
		printf("Wrong enter main command\n");
		if (status == EOF) break;
		continue;
	}

	if (strcmp(md.message, "create") == 0) {

		status = scanf("%d", &md.clientId);
		if (status <= 0) {
			printf("Wrong enter in block of create\n");
			continue;
		}

		if (socket_request == NULL) { // если дерево пусто вообще, то создаем тут
```

```
                context_request = zmq_ctx_new();
                socket_request = zmq_socket(context_request, ZMQ_PUSH);
                char tcp[100] {'\0'};
                sprintf(tcp, "%s%d", "tcp://127.0.0.1:400", md.clientId);
                int bind_sr = zmq_bind(socket_request, tcp); // создание сокета для ответов
                if (bind_sr == -1) {
                        perror("zmq_bind server");
                        zmq_close(socket_request);
                        zmq_ctx_destroy(context_request);
                        context_request = NULL;
                        socket_request = NULL;
                        continue;
                }
                int id = fork();
                if (id == -1) {
                        perror("fork error");
                        continue;
                } else if (id == 0) {
                        if (execl("../Consumer/a.out", tcp, (char*)&(md.clientId), NULL) == -
1) {
                                perror("execl");
                                continue;
                        }
                }
                child = md.clientId;
        } else { // иначе отправляем команду
                ZMQ_SEND(&md, socket_request);
        }

        //sleep(1);

    } else if (strcmp(md.message, "remove") == 0) {

        status = scanf("%d", &md.clientId);
        if (status <= 0) {
                printf("Wrong enter in block of remove\n");
                continue;
        }

        if (socket_request == NULL) {
                        printf("Error: Not found\n");
```

```
                continue;
        } else {
                ZMQ_SEND(&md, socket_request);
                if (child == md.clientId) {
                        ZMQ_SEND(&md, socket_request);
                        zmq_close(socket_request);
                        zmq_ctx_destroy(context_request);
                        socket_request = NULL;
                        context_request = NULL;
                        child = -1;
                }
        }


} else if (strcmp(md.message, "exec") == 0) {

        char endl1;
        status = scanf("%d%s%c", &md.clientId, md.name, &endl1);
        if (status <= 0 || status == 1) {
                printf("Wrong enter in block of exec\n");
                continue;
        } else if (endl1 == ' ') {
                status = scanf("%d", &md.value);
                if (status <= 0) {
                        printf("Wrong enter in block of exec\n");
                        continue;
                }

                if (socket_request == NULL) {
                        printf("Error: Not found\n");
                        continue;
                } else {
                        ZMQ_SEND(&md, socket_request);
                }
        } else if (endl1 == '\n') {
                if (socket_request == NULL) {
```

```
                printf("Error: Not found\n");
                continue;
            } else {
                ZMQ_SEND(&md, socket_request);
            }
        }

} else if (strcmp(md.message, "heartbit") == 0) {

        status = scanf("%d", &md.clientId);
        if (status <= 0) {
            printf("Wrong enter in block of heartbit\n");
            continue;
        }

        if (socket_request == NULL) {
            printf("Error: No nodes\n");
            continue;
        } else {
            ZMQ_SEND(&md, socket_request);
            last_heartbit_time = md.clientId;
        }
} else if (strcmp(md.message, "ping") == 0) {

        int id;
        status = scanf("%d", &id);
        if (status <= 0) {
            printf("Wrong enter in block of ping\n");
            continue;
        }

        auto it = times.find(id);
        if (it == times.end()) {
            printf("Ok: 0\n");
        } else {
            if ((time(NULL)-(it->second)) > last_heartbit_time * 4 ) {
                printf("Ok: 0\n");
            } else {
                printf("Ok: 1\n");
            }
        }
```

```
        } else {
                printf("Wrong command\n");
        }

}

if (socket_request != NULL)  {
        MessageData md;
        strcpy(md.message, "remove");
        md.clientId = child;
        ZMQ_SEND(&md, socket_request);
}

sleep(5);

start = false;
if (pthread_join(hand, NULL)) {
        perror("Поток не завершился. Завершение программы...");
   }

// We never get here though.
zmq_close(socket_answers);
zmq_ctx_destroy(context_answers);

zmq_close(socket_request);
zmq_ctx_destroy(context_request);

return 1;
}
```

client.cpp

```
#include <string.h>
#include <string>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>
#include <map>
#include "zmq.h"

typedef struct MD {
	char message[128];
	int clientId;
	char name[128] = {'\0'};
	int value = -1;
} MessageData;

bool ZMQ_SEND(MessageData *md, void* socket_request) {
	zmq_msg_t zmqMessage;
	zmq_msg_init_size(&zmqMessage, sizeof(MessageData));
	memcpy(zmq_msg_data(&zmqMessage), md, sizeof(MessageData));
	int send = zmq_msg_send(&zmqMessage, socket_request, ZMQ_DONTWAIT);
	if (send == -1) {
		perror("Zmq_msg_send error");
		zmq_msg_close(&zmqMessage);
		return false;
	}
	zmq_msg_close(&zmqMessage);
	return true;
}

bool start = true;
useconds_t timer = -1;
int id;

void* send_time(void* socket_answer) {

	while (start) {
		MessageData ok{"heartbit", id, "", id};
		ZMQ_SEND(&ok, socket_answer);
		usleep(timer);
	}
	return NULL;
}

int main(int argc, char const *argv[])
{
	void* my_pull = zmq_ctx_new();
	void* context_push_to_parrent = zmq_ctx_new();
```

```
        void* context_left_child = NULL;
        void* context_right_child = NULL;
        if (my_pull == NULL || context_push_to_parrent == NULL) perror("zmq_ctx_new error");

        void* my_socket = zmq_socket(my_pull, ZMQ_PULL);
        void* socket_answer = zmq_socket(context_push_to_parrent, ZMQ_PUSH);
        void* socket_left_child = NULL;
        void* socket_right_child = NULL;
        if (my_socket == NULL || socket_answer == NULL) perror("zmq_socket error");

        int bind_ms = zmq_connect(my_socket, argv[0]);
        int bind_sa = zmq_connect(socket_answer, "tcp://127.0.0.1:4004");
        if (bind_ms == -1 || bind_sa == -1) perror("zmq_bind client error");

        id = *(argv[1]);
        int children[2] {-1, -1};

        std::map<std::string, int> slov;

        pthread_t heart;

        //printf("Ok: %d\n", getpid());
        MessageData ok{"Ok: ", getpid(), "", id};
        ZMQ_SEND(&ok, socket_answer);

        for (;;)
        {

                zmq_msg_t message;
                zmq_msg_init(&message);
                zmq_msg_recv(&message, my_socket, 0);
                MessageData *md = (MessageData *)zmq_msg_data(&message);
                //printf("[%d] Message from producer: %s  clientId: %d name: %s value %d\n", id,
md->message, md->clientId, md->name, md->value);
                zmq_msg_close(&message);

                if (strcmp(md->message, "create") == 0) {

                        if (md->clientId == id) {
                                //printf("Error: Already exists\n");
```

```
                MessageData send{"Error: Already exists", -1, "", id};
                ZMQ_SEND(&send, socket_answer);
            } else if (md->clientId > id) {
                if (context_right_child == NULL) {
                    context_right_child = zmq_ctx_new();
                    socket_right_child = zmq_socket(context_right_child,
ZMQ_PUSH);
                    if (context_right_child == NULL || socket_right_child ==
NULL) {
                        perror("right child error");
                        continue;
                    }
                    char tcp[100] {'\0'};
                    sprintf(tcp, "%s%d", "tcp://127.0.0.1:400", md->clientId);
                    int bind_sr = zmq_bind(socket_right_child, tcp); // создание
сокета для ответов
                    if (bind_sr == -1) {
                        perror("zmq_bind server");
                        zmq_close(socket_right_child);
                        zmq_ctx_destroy(context_right_child);
                        context_right_child = NULL;
                        socket_right_child = NULL;
                        continue;
                    }
                    int id = fork();
                    if (id == -1) {
                        perror("fork error");
                        continue;
                    } else if (id == 0) {
                        if (execl("../Consumer/a.out", tcp, &md->clientId,
NULL) == -1) {
                            perror("execl");
                            continue;
                        }
                    }
                    children[1] = md->clientId;

                } else {
                    if (!ZMQ_SEND(md, socket_right_child)) {
                        zmq_close(socket_right_child);
                        zmq_ctx_destroy(context_right_child);
                        context_right_child = NULL;
                        socket_right_child = NULL;
                        children[1] = -1;
                    }
                }
```

```
                } else {
                        if (context_left_child == NULL) {
                                context_left_child = zmq_ctx_new();
                                socket_left_child = zmq_socket(context_left_child,
ZMQ_PUSH);
                                if (context_left_child == NULL || socket_left_child == NULL)
{
                                        perror("right child error");
                                        continue;
                                }
                                char tcp[100] {'\0'};
                                sprintf(tcp, "%s%d", "tcp://127.0.0.1:400", md->clientId);
                                int bind_sr = zmq_bind(socket_left_child, tcp); // создание
сокета для ответов
                                if (bind_sr == -1) {
                                        perror("zmq_bind server error");
                                        zmq_close(socket_left_child);
                                        zmq_ctx_destroy(context_left_child);
                                        context_left_child = NULL;
                                        socket_left_child = NULL;
                                        continue;
                                }
                                int id = fork();
                                if (id == -1) {
                                        perror("fork error");
                                        continue;
                                } else if (id == 0) {
                                        if (execl("../Consumer/a.out", tcp, &md->clientId,
NULL) == -1) {
                                                perror("execl error");
                                                continue;
                                        }
                                }
                                children[0] = md->clientId;
                        } else {
                                //ZMQ_SEND(md, socket_left_child);
                                if (!ZMQ_SEND(md, socket_left_child)) {
                                        zmq_close(socket_left_child);
                                        zmq_ctx_destroy(context_left_child);
                                        context_left_child = NULL;
```

```
                        socket_left_child = NULL;
                        children[0] = -1;
                    }
                }
            }
            //printf("[%d] left - %d, right - %d\n", id, children[0], children[1]);

        } else if (strcmp(md->message, "remove") == 0) {

            if (md->clientId == id) {
                if (children[0] != -1) {
                    md->clientId = children[0];
                    ZMQ_SEND(md, socket_left_child);
                }
                if (children[1] != -1) {
                    md->clientId = children[1];
                    ZMQ_SEND(md, socket_right_child);
                }
                //printf("Ok\n");
                MessageData send{"Ok", -1, "", id};
                ZMQ_SEND(&send, socket_answer);
                break;
            } else if (md->clientId > id) {
                if (context_right_child == NULL) {
                    //printf("Error: Not found\n");
                    MessageData send{"Error: Not found", -1, "", id};
                    ZMQ_SEND(&send, socket_answer);
                } else {
                    if (!ZMQ_SEND(md, socket_right_child)) {
                        zmq_close(socket_right_child);
                        zmq_ctx_destroy(context_right_child);
                        context_right_child = NULL;
                        socket_right_child = NULL;
                        children[1] = -1;
                    }
                    if (children[1] == md->clientId) {
                        zmq_close(socket_right_child);
                        zmq_ctx_destroy(context_right_child);
                        socket_right_child = NULL;
                        context_right_child = NULL;
                        children[1] = -1;
                    }
                }
            } else {
                if (context_left_child == NULL) {
                    //printf("Error: Not found\n");
```

```
                                MessageData send{"Error: Not found", -1, "", id};
                                ZMQ_SEND(&send, socket_answer);
                        } else {
                                if (!ZMQ_SEND(md, socket_left_child)) {
                                        zmq_close(socket_left_child);
                                        zmq_ctx_destroy(context_left_child);
                                        context_left_child = NULL;
                                        socket_left_child = NULL;
                                        children[0] = -1;
                                }
                                if (children[0] == md->clientId) {
                                        zmq_close(socket_left_child);
                                        zmq_ctx_destroy(context_left_child);
                                        socket_left_child = NULL;
                                        context_left_child = NULL;
                                        children[0] = -1;
                                }
                        }
                }

        } else if (strcmp(md->message, "exec") == 0) {

                if (md->clientId == id) {
                        if (md->value == -1) {
                                auto it = slov.find(std::string(md->name));
                                if (it == slov.end()) {
                                        //printf("Ok:%d: %s not found\n", id, md->name);
                                        MessageData send{"Ok:", id, "", id};
                                        //send.name = md->name;
                                        strcpy(send.name, md->name);
                                        ZMQ_SEND(&send, socket_answer);
                                } else {
                                        //printf("Ok:%d\n", it->second);
                                        MessageData send{"Ok:", it->second, "", id};
                                        ZMQ_SEND(&send, socket_answer);
                                }
                        } else {
                                slov.insert(std::pair<std::string, int>(std::string(md->name),
md->value));
                                //printf("Ok:%d\n", id);
```

```
                MessageData send{"Ok:", id, "", id};
                ZMQ_SEND(&send, socket_answer);
            }
        } else if (md->clientId > id) {
            if (context_right_child == NULL) {
                //printf("Error: Not found\n");
                MessageData send{"Error: Not found", -1, "", id};
                ZMQ_SEND(&send, socket_answer);
            } else {
                if (!ZMQ_SEND(md, socket_right_child)) {
                    zmq_close(socket_right_child);
                    zmq_ctx_destroy(context_right_child);
                    context_right_child = NULL;
                    socket_right_child = NULL;
                    children[1] = -1;
                }
            }
        } else {
            if (context_left_child == NULL) {
                //printf("Error: Not found\n");
                MessageData send{"Error: Not found", -1, "", id};
                ZMQ_SEND(&send, socket_answer);
            } else {
                if (!ZMQ_SEND(md, socket_left_child)) {
                    zmq_close(socket_left_child);
                    zmq_ctx_destroy(context_left_child);
                    context_left_child = NULL;
                    socket_left_child = NULL;
                    children[0] = -1;
                }
            }
        }

    } else if (strcmp(md->message, "heartbit") == 0) {
        if (children[0] != -1) {
            if (!ZMQ_SEND(md, socket_left_child)) {
                zmq_close(socket_left_child);
                zmq_ctx_destroy(context_left_child);
                context_left_child = NULL;
                socket_left_child = NULL;
                children[0] = -1;
            }
        }
        if (children[1] != -1) {
            if (!ZMQ_SEND(md, socket_right_child)) {
                zmq_close(socket_right_child);
```

```
                                zmq_ctx_destroy(context_right_child);
                                context_right_child = NULL;
                                socket_right_child = NULL;
                                children[1] = -1;
                        }
                }

                timer = md->clientId;
                start = false;
                sleep(1);
                start = true;
                if (pthread_create(&heart, NULL, send_time, socket_answer) != 0)
perror("Поток не смог создаться");
                MessageData send{"Ok", -1, "", id};
                ZMQ_SEND(&send, socket_answer);

                continue;

        }
    }


    zmq_close(my_socket);
    zmq_ctx_destroy(my_pull);
    zmq_close(socket_answer);
    zmq_ctx_destroy(context_push_to_parrent);
    zmq_close(socket_left_child);
    zmq_ctx_destroy(context_left_child );
    zmq_close(socket_right_child);
    zmq_ctx_destroy(context_right_child);

    start = false;
    pthread_join(heart, NULL);

    return 0;
}
```

Тесты и протокол исполнения

igor@igor-Aspire-A315-53G:~/Рабочий стол/FireAndForget/Producer$ ./a.out
Producer starting...
create 8
Ok: 77018
create 3
Ok: 77027
create 5
Ok: 77036
create 16
Ok: 77064
create 13
Ok: 77317
create 15
Ok: 77324
exec 13 abc 30
Ok:13
exec 13 abcd
Ok:13: 'abcd' not found
exec 13 abc
Ok:30
exec 8 abc
Ok:8: 'abc' not found
remove 5
Ok
remove 16
Ok
Ok
Ok
create 15
Ok: 77346
create 16
Ok: 77353
heartbit 60
Ok
Ok
Ok
Ok
ping 15
Ok: 1
ping 8
Ok: 1
ping 10
Ok: 0

Выводы

Данная лабораторная работа оказалась самой сложной по сравнению со всеми предыдущими. Я научился использовать локальную сеть компьютера для передачи сообщений между процессами. Организовал топологию своего варианта, что хорошо прослеживается по коду. Интересно было то, что NULL ссылки в бинарном дереве очень хорошо отождествлялись с сокетами. Использовал знания о потоках, чтобы реализовать команду heartbit. Познал ZMQ. К сожалению, иногда все же бывают некоторые поломки в программе, однако система все равно продолжает функционировать, хоть и не на полную мощность. В дальнейшем попробую написать различные программы с очередями сообщений, только на языке python, чтобы лучше понять принципы работы.

Список литературы

1. Поисковик Google [электронный ресурс] URL: https://google.com/ (дата обращения: 11.12.2020)
2. Таненбаум Э., Бос Х. *Современные операционные системы. — 4-е изд.* — СПб.: Издательский дом «Питер», 2018. — С. 111 - 123