Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

Группа: М80-206Б-20

Студент: Кочев Д.О.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 29.12.2023

Москва, 2023

**Постановка задачи**

**Вариант 33.**

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:

Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не

удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5

Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Топология 2

Аналогично топологии 2, но узлы находятся в дереве общего вида.

Исполнение команды на вычислительном узле

Набор команд 1 (подсчет суммы n чисел)

Формат команды: exec id n k1 … kn

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

n – количество складываемых чисел (от 1 до 108)

k1 … kn – складываемые числа

Пример:

> exec 10 3 1 2 3

Ok:10: 6

Тип проверки доступности узлов

Команда проверки 3

Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую.

Пример:

> pingall

Ok: -1 // Все узлы доступны

> pingall

Ok: 7;10;15 // узлы 7, 10, 15 — недоступны

**Общий метод и алгоритм решения**

Программа состоит из нескольких компонентов, включая вычислительные, и один управляющий узел. При создании нового вычислительного узла, который добавится в дерево, будет запущен новый процесс, при этом родителем нового процесса будет тот, кого указал пользователь. Взаимодействие между узлами будет осуществляться с использованием брокера сообщений, в моем случае ZMQ. Разработан интерфейс командной строки для интерактивного общения с пользователем. Если мы удалим один из узлов, то также будут удалены все его потомки. Это мы обозначим в своем дереве.

**Код программы**

**client.cpp**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <zmq.h> // gcc client.c -o client -lzmq

int id;

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

// id = atoi(argv[0]);

sscanf(argv[1], "%d", &id);

printf("Ok my id is: %d\n", id);

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

void \*subscriber = zmq\_socket(context, ZMQ\_SUB); // Сокет для принятия сообщений

zmq\_connect(subscriber, "tcp://127.0.0.1:5555"); // Подключаемся к адресу

zmq\_setsockopt(subscriber, ZMQ\_SUBSCRIBE, "", 0); // Подписываемся на все сообщения (пустая строка)

while(1){

char buffer[1024]; // Буфер для принятого сообщения

char command[20]; // хранение команды

int arg1, arg2;

memset(buffer, 0, sizeof(buffer)); // очищаем buffer

memset(command, 0, sizeof(command)); // очищаем buffer

zmq\_recv(subscriber, buffer, sizeof(buffer), 0); // Принятие сообщения

// printf("message: %s\n", buffer);

sscanf(buffer, "%s", command); // Считываем начальное слово в command

if (strcmp(command, "create") == 0){

sscanf(buffer, "%\*s %d %d", &arg1, &arg2);

if (id == arg2){ // если команда предназначена для нас, то выполняем

printf("Node %d: create child\n", id);

pid\_t id\_child = fork();

if (id\_child == 0){

char str1[sizeof(int)];

sprintf(str1, "%d", arg1);

execl("./client", "./client", str1, NULL);

perror("execl");

}

printf("Ok: %d\n", id\_child);

}

}

else if (strcmp(command, "exec") == 0){

sscanf(buffer, "%\*s %d %d", &arg1, &arg2);

if (id == arg1){ // если команда предназначена для нас, то выполняем

int\* array;

array = (int\*)malloc(arg2 \* sizeof(int));

int i = 0;

char \*ptr = strchr(buffer, ' ');

// читаем массив чисел из строки (+2, потому что мы в этот массив читаем еще первые два аргумента из buffer, то есть arg1 и arg2)

while (ptr != NULL && sscanf(ptr, "%d", &array[i]) == 1 && i != arg2 + 2) {

i++;

ptr = strchr(ptr + 1, ' ');

}

int s = 0;

for (int i = 2; i < arg2 + 2; i++){ // складываем числа

s += array[i];

}

printf("Ok:%d: %d\n", id, s);

free(array);

}

}

else if(strcmp(command, "kill") == 0){

sscanf(buffer, "%\*s %d", &arg1);

if (id == arg1){

printf("Node %d: kill myself\n", id);

break;

}

}

if (strcmp(command, "killall") == 0){ // когда строка равна "killall"

sscanf(buffer, "%\*s %d", &arg1);

if (arg1 == 1234) // пароль :)

break;

}

}

zmq\_close(subscriber);

zmq\_ctx\_destroy(context);

}

**server.cpp**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <mqueue.h>

#include <zmq.h> // gcc server.c -o server -lzmq

#include "tree.h" // strace -f

char message[1000000];

void killWithChildren(TNode \*node, void \* publisher) { // удаляем процессы

if (node == NULL) {

return;

}

// Отключаем текущий узел

sprintf(message, "kill %d", node->data);

zmq\_send(publisher, message, strlen(message), 0);

memset(message, 0, sizeof(message));

// Рекурсивно отключаем всех потомков

killWithChildren(node->firstChild, publisher);

killWithChildren(node->nextBrother, publisher);

}

int pingallCommand(TNode \*node, int i){

if (node == NULL) {

return i;

}

if (node->exist == false){

printf("%d;", node->data);

i += 1;

}

// Рекурсивно отключаем всех потомков

i = pingallCommand(node->firstChild, i);

i = pingallCommand(node->nextBrother, i);

return i;

}

int main(){

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

void \*publisher = zmq\_socket(context, ZMQ\_PUB); // Сокет для отправки сообщений

zmq\_bind(publisher, "tcp://127.0.0.1:5555"); // Привязываем сокет к адресу

char input[256];

char command[60]; // массив для ввода команды

// pid\_t \* my\_forks = (pid\_t\*)malloc(100 \* sizeof(pid\_t)); // 100 дочерних процессов

int count\_of\_forks = 0;

int param1, param2;

TNode \*root = createNode(-1);

while(1){

memset(message, 0, sizeof(message)); // очищаем строку сообщения

memset(command, 0, sizeof(command)); // очищаем command

// memset(input, 0, sizeof(input)); // очищаем input

if (fgets(input, sizeof(input), stdin) == NULL) { // Считываем вводную строку (NULL)

// printf("adios\n");

// Если встречен конец файла, завершаем цикл

break;

}

sscanf(input, "%s", command); //читаем команду

if (strcmp(command, "create") == 0){ // ------------------------------------create----------------------------------------

sscanf(input, "%\*s %d %d", &param1, &param2);

TNode \*node = find\_node(root, param2); //ищем родителя

TNode \*node2 = find\_node(root, param1); //ищем id, вдруг он уже есть

if (node == NULL){

printf("Error: Parent not found \n");

// return 1;

}

else if(node2 != NULL){

printf("Error: Already exist \n");

// return 1;

}

else{

addChild(node, param1); //добавляем дочерний процесс в дерево

if (param2 == -1){ // если родительский процесс - родитель

pid\_t id = fork();

if (id == 0){

char str1[sizeof(int)];

sprintf(str1, "%d", param1);

execl("./client", "./client", str1, NULL);

perror("execl");

}

printf("Ok: %d\n", id);

// my\_forks[count\_of\_forks] = id;

// count\_of\_forks ++;

}

else{ // если родительский процесс - кто-то в дереве

sprintf(message, "create %d %d", param1, param2); // создаем строку message

zmq\_send(publisher, message, strlen(message), 0);

}

}

}

else if (strcmp(command, "exec") == 0){ // --------------------exec--------------------------------

sscanf(input, "%\*s %d", &param1);

TNode \*node3 = find\_node(root, param1);

if (param1 == -1)

printf("The controlling process only controls\n");

else if(node3 == NULL){

printf("This node is dead or doesn`t exist ever\n");

}

else{

zmq\_send(publisher, input, strlen(input), 0);

}

}

else if (strcmp(command, "kill") == 0){

sscanf(input, "%\*s %d", &param1);

TNode \*node4 = find\_node(root, param1);

if (param1 == -1)

printf("Nope\n");

else if(node4 == NULL){

printf("This node is dead or doesn`t exist ever\n");

}

else{

zmq\_send(publisher, input, strlen(input), 0);

killWithChildren(node4->firstChild, publisher);

disableNode(node4->firstChild);

disableOneNode(node4);

}

}

else if(strcmp(command, "pingall") == 0){

printf("Ok:");

int i = pingallCommand(root, 0);

if (i == 0)

printf("-1");

printf("\n");

}

}

const char \*message2 = "killall 1234";

zmq\_send(publisher, message2, strlen(message2), 0);

printTree(root, 0);

freeTree(root);

// free(my\_forks);

zmq\_close(publisher);

zmq\_ctx\_destroy(context);

}

**tree.h**

#include <stdio.h>

#include <stdbool.h>

typedef struct TNode {

int data;

struct TNode \*firstChild; // указатель на первого потомка

struct TNode \*nextBrother; // указатель на следующего брата

bool exist;

} TNode;

TNode \*createNode(int data) {

TNode \*newNode = (TNode \*)malloc(sizeof(TNode));

if (newNode != NULL) {

newNode->data = data;

newNode->firstChild = NULL;

newNode->nextBrother = NULL;

newNode->exist = true;

}

return newNode;

}

void addChild(TNode \*parent, int data) {

TNode \*newChild = createNode(data);

if (newChild == NULL) {

fprintf(stderr, "Failed to create a new child node.\n");

return;

}

// Если у узла нет потомков, добавляем нового потомка

if (parent->firstChild == NULL) {

parent->firstChild = newChild;

} else {

// Если у узла уже есть потомок, идем по списку братьев

TNode \*brother = parent->firstChild;

while (brother->nextBrother != NULL) {

brother = brother->nextBrother;

}

brother->nextBrother = newChild;

}

}

// TreeNode \*root = createNode(1);

// addChild(root, 2);

int findPath(TNode \*currentNode, int targetData, int \*path, int depth) { // дорога до узла с нужным значением

if (currentNode == NULL) {

return 0;

}

// Если текущий узел содержит искомые данные, завершаем рекурсию

if (currentNode->data == targetData) {

path[depth] = -1; // 0 означает, что текущий узел содержит искомые данные

return 1;

}

// Пытаемся найти путь в потомках

int childIndex = 1;

if (findPath(currentNode->firstChild, targetData, path, depth + 1)) {

path[depth] = childIndex;

return 1;

}

// Пытаемся найти путь в братьях

if (findPath(currentNode->nextBrother, targetData, path, depth)) {

path[depth] = childIndex;

return 1;

}

return 0; // Искомый узел не найден

}

// int targetData = 6;

// int path[100]; // здесь будет сохранен путь

// int found = findPath(root, targetData, path, 0);

// Функция для удаления узла и всех его потомков

void deleteNodeAndChildren(TNode \*parent, TNode \*toDelete) {

if (parent == NULL || toDelete == NULL) {

return;

}

// Если удаляемый узел - первый потомок родителя

if (parent->firstChild == toDelete) {

parent->firstChild = toDelete->nextBrother;

} else {

// Если удаляемый узел - брат

TNode \*brother = parent->firstChild;

while (brother != NULL && brother->nextBrother != toDelete) {

brother = brother->nextBrother;

}

if (brother != NULL) {

brother->nextBrother = toDelete->nextBrother;

}

}

// Обновляем указатель на следующего брата у левого брата удаляемого узла

if (toDelete->nextBrother != NULL && parent->firstChild != toDelete) {

TNode \*leftBrother = parent->firstChild;

while (leftBrother->nextBrother != toDelete) {

leftBrother = leftBrother->nextBrother;

}

leftBrother->nextBrother = toDelete->nextBrother;

}

// Рекурсивно удаляем всех потомков узла

TNode \*child = toDelete->firstChild;

while (child != NULL) {

TNode \*nextChild = child->nextBrother;

deleteNodeAndChildren(toDelete, child);

child = nextChild;

}

free(toDelete);

}

// Функция для поиска узла по значению

TNode \*findNodeAndParent(TNode \*root, int targetData, TNode \*\*parent) {

if (root == NULL) {

return NULL;

}

if (root->data == targetData) {

return root;

}

// Рекурсивный вызов для потомков

TNode \*result = findNodeAndParent(root->firstChild, targetData, &root);

if (result != NULL) {

\*parent = root;

return result;

}

// Рекурсивный вызов для братьев

return findNodeAndParent(root->nextBrother, targetData, parent);

}

// int targetData = 4;

// TreeNode \*parent = NULL;

// TreeNode \*toFind = findNodeAndParent(root, targetData, &parent);

TNode\* find\_node(TNode \*root, int data) {

if (root == NULL) {

return NULL; // Дошли до конца поддерева, узел не найден

}

if (root->data == data && root->exist == true) { // root->exist == true убрать и тогда нельзя создавать прошлые

return root; // Узел найден

}

// Рекурсивно ищем в потомках и братьях

TNode \*found\_in\_child = find\_node(root->firstChild, data);

if (found\_in\_child != NULL) {

return found\_in\_child; // Найден в потомках

}

return find\_node(root->nextBrother, data); // Ищем в братьях

}

void freeTree(TNode \*root) {

if (root == NULL) {

return; // Базовый случай: пустое поддерево

}

// Рекурсивный вызов для всех потомков текущего узла

freeTree(root->firstChild);

// Рекурсивный вызов для всех братьев текущего узла

freeTree(root->nextBrother);

// Освобождение памяти для текущего узла

free(root);

}

void printTree(TNode\* root, int depth) { // глубина всегда в начале 0

if (root == NULL) {

return;

}

// Вывод текущего узла с отступом, зависящим от глубины в дереве

for (int i = 0; i < depth; ++i) {

printf(" ");

}

printf("%d", root->data);

if (root->exist == false)

printf("k");

printf("\n");

// Рекурсивный вызов для всех потомков текущего узла

printTree(root->firstChild, depth + 1);

// Рекурсивный вызов для всех братьев текущего узла

printTree(root->nextBrother, depth);

}

void disableNode(TNode \*node) { // отключаем узел

if (node == NULL) {

return;

}

// Отключаем текущий узел

node->exist = false;

// Рекурсивно отключаем всех потомков

disableNode(node->firstChild);

disableNode(node->nextBrother);

}

void disableOneNode(TNode \*node) {

node->exist = false;

}

**Протокол работы программы**

**Тестирование:**

$ ./server

create 2-1

Ok: 101143

Ok my id is: 2

create 3 2

Node 2: create child

Ok: 101182

Ok my id is: 3

create 4 3

Node 3: create child

Ok: 101206

Ok my id is: 4

create 5 4

Node 4: create child

Ok: 101257

Ok my id is: 5

exec 4 2 7 8

Ok:4: 15

kill 3

Node 5: kill myself

Node 4: kill myself

Node 3: kill myself

pingall

Ok:3;4;5;

-1

2

3k

4k

5k

**Часть Strace, полный в strace.txt:**

$ strace -f ./server

execve("./server", ["./server"], 0x7fff026f4d78 /\* 36 vars \*/) = 0

brk(NULL) = 0x56242bd13000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe646ffdc0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff096387000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=19839, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 19839, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7ff096382000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libzmq.so.5", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\233\1\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=634936, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 636784, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff0962e6000

mmap(0x7ff0962fe000, 397312, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7ff0962fe000

mmap(0x7ff09635f000, 106496, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x79000) = 0x7ff09635f000

mmap(0x7ff096379000, 36864, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x92000) = 0x7ff096379000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0 =\340\2563\265?\356\25x\261\27\313A#\350"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, AT\_EMPTY\_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff0960be000

mmap(0x7ff0960e6000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7ff0960e6000

mmap(0x7ff09627b000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7ff09627b000

mmap(0x7ff0962d3000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7ff0962d3000

mmap(0x7ff0962d9000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff0962d9000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libbsd.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=89096, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 94432, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff0960a6000

mprotect(0x7ff0960aa000, 69632, PROT\_NONE) = 0

mmap(0x7ff0960aa000, 53248, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7ff0960aa000

mmap(0x7ff0960b7000, 12288, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x11000) = 0x7ff0960b7000

mmap(0x7ff0960bb000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x14000) = 0x7ff0960bb000

mmap(0x7ff0960bd000, 224, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff0960bd000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libsodium.so.23", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=355040, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 357440, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff09604e000

mprotect(0x7ff09605a000, 303104, PROT\_NONE) = 0

mmap(0x7ff09605a000, 229376, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xc000) = 0x7ff09605a000

mmap(0x7ff096092000, 69632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x44000) = 0x7ff096092000

mmap(0x7ff0960a4000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x55000) = 0x7ff0960a4000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpgm-5.3.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340L\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=310264, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 329808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff095ffd000

mmap(0x7ff096001000, 172032, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7ff096001000

mmap(0x7ff09602b000, 118784, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2e000) = 0x7ff09602b000

mmap(0x7ff096048000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4a000) = 0x7ff096048000

mmap(0x7ff09604a000, 14416, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff09604a000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libnorm.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \255\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=497824, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff095ffb000

mmap(NULL, 1223168, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff095ed0000

mprotect(0x7ff095eda000, 446464, PROT\_NONE) = 0

mmap(0x7ff095eda000, 286720, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa000) = 0x7ff095eda000

mmap(0x7ff095f20000, 155648, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x50000) = 0x7ff095f20000

mmap(0x7ff095f47000, 16384, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x76000) = 0x7ff095f47000

mmap(0x7ff095f4b000, 719360, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff095f4b000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgssapi\_krb5.so.2", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=338648, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 340960, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff095e7c000

mprotect(0x7ff095e87000, 282624, PROT\_NONE) = 0

mmap(0x7ff095e87000, 229376, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xb000) = 0x7ff095e87000

mmap(0x7ff095ebf000, 49152, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x43000) = 0x7ff095ebf000

mmap(0x7ff095ecc000, 16384, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4f000) = 0x7ff095ecc000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=2260296, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2275520, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff095c50000

mprotect(0x7ff095cea000, 1576960, PROT\_NONE) = 0

mmap(0x7ff095cea000, 1118208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x9a000) = 0x7ff095cea000

mmap(0x7ff095dfb000, 454656, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ab000) = 0x7ff095dfb000

mmap(0x7ff095e6b000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x21a000) = 0x7ff095e6b000

mmap(0x7ff095e79000, 10432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff095e79000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=125488, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 127720, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff095c30000

mmap(0x7ff095c33000, 94208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7ff095c33000

mmap(0x7ff095c4a000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a000) = 0x7ff095c4a000

mmap(0x7ff095c4e000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1d000) = 0x7ff095c4e000

close(3) = 0

…

[pid 106407] rseq(0x7ff095201fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 106405] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 106407] <... rseq resumed>) = 0

[pid 106405] eventfd2(0, EFD\_CLOEXEC <unfinished ...>

[pid 106407] set\_robust\_list(0x7ff095201920, 24 <unfinished ...>

[pid 106405] <... eventfd2 resumed>) = 8

[pid 106407] <... set\_robust\_list resumed>) = 0

[pid 106405] fcntl(8, F\_GETFL <unfinished ...>

[pid 106407] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 106405] <... fcntl resumed>) = 0x2 (flags O\_RDWR)

[pid 106407] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 106405] fcntl(8, F\_SETFL, O\_RDWR|O\_NONBLOCK <unfinished ...>

[pid 106407] rt\_sigprocmask(SIG\_BLOCK, ~[RTMIN RT\_1], <unfinished ...>

[pid 106405] <... fcntl resumed>) = 0

[pid 106407] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 106405] fcntl(8, F\_GETFL <unfinished ...>

[pid 106407] sched\_getparam(106407, <unfinished ...>

[pid 106405] <... fcntl resumed>) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

[pid 106407] <... sched\_getparam resumed>[0]) = 0

[pid 106405] fcntl(8, F\_SETFL, O\_RDWR|O\_NONBLOCK <unfinished ...>

[pid 106407] sched\_getscheduler(106407 <unfinished ...>

[pid 106405] <... fcntl resumed>) = 0

[pid 106407] <... sched\_getscheduler resumed>) = 0 (SCHED\_OTHER)

[pid 106405] getpid( <unfinished ...>

[pid 106407] sched\_setscheduler(106407, SCHED\_OTHER, [0] <unfinished ...>

[pid 106405] <... getpid resumed>) = 106405

[pid 106407] <... sched\_setscheduler resumed>) = 0

[pid 106405] getpid() = 106405

[pid 106407] prctl(PR\_SET\_NAME, "ZMQbg/IO/0" <unfinished ...>

[pid 106405] poll([{fd=8, events=POLLIN}], 1, 0 <unfinished ...>

[pid 106407] <... prctl resumed>) = 0

[pid 106405] <... poll resumed>) = 0 (Timeout)

[pid 106405] socket(AF\_NETLINK, SOCK\_RAW|SOCK\_CLOEXEC, NETLINK\_ROUTE) = 9

[pid 106405] bind(9, {sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, 12) = 0

[pid 106407] epoll\_wait(7, <unfinished ...>

[pid 106405] getsockname(9, {sa\_family=AF\_NETLINK, nl\_pid=106405, nl\_groups=00000000}, [12]) = 0

[pid 106405] sendto(9, [{nlmsg\_len=20, nlmsg\_type=RTM\_GETLINK, nlmsg\_flags=NLM\_F\_REQUEST|NLM\_F\_DUMP, nlmsg\_seq=1702686593, nlmsg\_pid=0}, {ifi\_family=AF\_UNSPEC, ...}], 20, 0, {sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, 12) = 20

[pid 106405] recvmsg(9, {msg\_name={sa\_family=AF\_NETLINK, nl\_pid=0, nl\_groups=00000000}, msg\_namelen=12, msg\_iov=[{iov\_base=[[{nlmsg\_len=1336, nlmsg\_type=RTM\_NEWLINK, nlmsg\_flags=NLM\_F\_MULTI, nlmsg\_seq=1702686593, nlmsg\_pid=106405}, {ifi\_family=AF\_UNSPEC, ifi\_type=ARPHRD\_LOOPBACK, ifi\_index=if\_nametoindex("lo"), ifi\_flags=IFF\_UP|IFF\_LOOPBACK|IFF\_RUNNING|IFF\_LOWER\_UP, ifi\_change=0}, [[{nla\_len=7, nla\_type=IFLA\_IFNAME}, "lo"], [{nla\_len=8, nla\_type=IFLA\_TXQLEN}, 1000], type=NLMSG\_DONE, nlmsg\_flags=NLM\_F\_MULTI, nlmsg\_seq=1702686594, nlmsg\_pid=106405}, 0], iov\_len=4096}], msg\_iovlen=1, msg\_controllen=0, msg\_flags=0}, 0) = 20

[pid 106405] close(9) = 0

[pid 106405] socket(AF\_INET, SOCK\_STREAM|SOCK\_CLOEXEC, IPPROTO\_TCP) = 9

[pid 106405] setsockopt(9, SOL\_SOCKET, SO\_REUSEADDR, [1], 4) = 0

[pid 106405] bind(9, {sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("127.0.0.1")}, 16) = 0

[pid 106405] listen(9, 100) = 0

[pid 106405] getsockname(9, {sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("127.0.0.1")}, [128 => 16]) = 0

[pid 106405] getsockname(9, {sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("127.0.0.1")}, [128 => 16]) = 0

[pid 106405] getpid() = 106405

[pid 106405] write(6, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106407] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=735224416, u64=94713354035808}}], 256, -1) = 1

[pid 106405] getpid( <unfinished ...>

[pid 106407] getpid( <unfinished ...>

[pid 106405] <... getpid resumed>) = 106405

[pid 106407] <... getpid resumed>) = 106405

[pid 106405] write(8, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

[pid 106405] <... write resumed>) = 8

[pid 106407] <... poll resumed>) = 1 ([{fd=6, revents=POLLIN}])

[pid 106407] getpid() = 106405

[pid 106407] read(6, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106405] newfstatat(0, "", <unfinished ...>

[pid 106407] mmap(NULL, 134217728, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_NORESERVE, -1, 0 <unfinished ...>

[pid 106405] <... newfstatat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x4), ...}, AT\_EMPTY\_PATH) = 0

[pid 106407] <... mmap resumed>) = 0x7ff08ca01000

[pid 106405] read(0, <unfinished ...>

[pid 106407] munmap(0x7ff08ca01000, 56619008) = 0

[pid 106407] munmap(0x7ff094000000, 10489856) = 0

[pid 106407] mprotect(0x7ff090000000, 135168, PROT\_READ|PROT\_WRITE) = 0

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_ADD, 9, {events=0, data={u32=2415922032, u64=140671184800624}}) = 0

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN, data={u32=2415922032, u64=140671184800624}}) = 0

[pid 106407] getpid() = 106405

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0) = 0 (Timeout)

[pid 106407] epoll\_wait(7, create 2 -1

<unfinished ...>

…

[pid 106405] <... read resumed>"create 2 -1\n", 1024) = 12

[pid 106405] clone(child\_stack=NULL, sin\_addr=inet\_addr("127.0.0.1")}, [128 => 16]) = 0

[pid 106627] epoll\_ctl(7, EPOLL\_CTL\_DEL, 9, 0x7f2ba8001434 <unfinished ...>

[pid 106407] fcntl(11, F\_GETFL <unfinished ...>

[pid 106627] <... epoll\_ctl resumed>) = 0

[pid 106407] <... fcntl resumed>) = 0x2 (flags O\_RDWR)

[pid 106627] getsockopt(9, SOL\_SOCKET, SO\_ERROR, <unfinished ...>

[pid 106407] fcntl(11, F\_SETFL, O\_RDWR|O\_NONBLOCK <unfinished ...>

[pid 106627] <... getsockopt resumed>[0], [4]) = 0

[pid 106407] <... fcntl resumed>) = 0

[pid 106627] setsockopt(9, SOL\_TCP, TCP\_NODELAY, [1], 4 <unfinished ...>

[pid 106407] getpid( <unfinished ...>

[pid 106627] <... setsockopt resumed>) = 0

[pid 106407] <... getpid resumed>) = 106405

[pid 106627] getsockname(9, <unfinished ...>

[pid 106407] write(6, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

[pid 106627] <... getsockname resumed>{sa\_family=AF\_INET, sin\_port=htons(59230), sin\_addr=inet\_addr("127.0.0.1")}, [128 => 16]) = 0

[pid 106407] <... write resumed>) = 8

[pid 106627] getpeername(9, <unfinished ...>

[pid 106407] epoll\_wait(7, <unfinished ...>

[pid 106627] <... getpeername resumed>{sa\_family=AF\_INET, sin\_port=htons(5555), sin\_addr=inet\_addr("127.0.0.1")}, [128 => 16]) = 0

[pid 106407] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=735224416, u64=94713354035808}}], 256, -1) = 1

[pid 106627] fcntl(9, F\_GETFL <unfinished ...>

[pid 106407] getpid( <unfinished ...>

[pid 106627] <... fcntl resumed>) = 0x802 (flags O\_RDWR|O\_NONBLOCK)

[pid 106407] <... getpid resumed>) = 106405

[pid 106627] fcntl(9, F\_SETFL, O\_RDWR|O\_NONBLOCK <unfinished ...>

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

[pid 106627] <... fcntl resumed>) = 0

[pid 106407] <... poll resumed>) = 1 ([{fd=6, revents=POLLIN}])

data={u32=2415988864, u64=140671184867456}} <unfinished ...>

[pid 106627] <... sendto resumed>) = 27

[pid 106407] <... epoll\_ctl resumed>) = 0

[pid 106627] epoll\_wait(7, <unfinished ...>

[pid 106407] recvfrom(11, <unfinished ...>

[pid 106627] <... epoll\_wait resumed>[{events=EPOLLOUT, data={u32=2818577456, u64=139825478898736}}], 256, 29996) = 1

[pid 106407] <... recvfrom resumed>"\4\31\5READY\vSocket-Type\0\0\0\3SUB", 8192, 0, NULL, NULL) = 27

[pid 106407] epoll\_wait(7, [{events=EPOLLOUT, data={u32=2415988864, u64=140671184867456}}], 256, 29993) = 1

[pid 106407] mprotect(0x7ff090021000, 4096, PROT\_READ|PROT\_WRITE) = 0

[pid 106407] getpid() = 106405

[pid 106407] write(8, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106407] sendto(11, "\4\31\5READY\vSocket-Type\0\0\0\3PUB", 27, 0, NULL, 0) = 27

[pid 106407] epoll\_wait(7, [{events=EPOLLOUT, data={u32=2415988864, u64=140671184867456}}], 256, -1) = 1

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_MOD, 11, {events=EPOLLIN, data={u32=2415988864, u64=140671184867456}}) = 0

[pid 106407] epoll\_wait(7, <unfinished ...>

[pid 106627] epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN, data={u32=2818577456, u64=139825478898736}}) = 0

[pid 106627] epoll\_wait(7, [{events=EPOLLIN, data={u32=2818577456, u64=139825478898736}}], 256, 29990) = 1

[pid 106627] recvfrom(9, "\4\31\5READY\vSocket-Type\0\0\0\3PUB", 8192, 0, NULL, NULL) = 27

[pid 106627] epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN|EPOLLOUT, data={u32=2818577456, u64=139825478898736}}) = 0

[pid 106627] sendto(9, "\4\n\tSUBSCRIBE", 12, 0, NULL, 0) = 12

[pid 106407] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=2415988864, u64=140671184867456}}], 256, -1) = 1

[pid 106627] epoll\_wait(7, <unfinished ...>

[pid 106407] recvfrom(11, <unfinished ...>

[pid 106627] <... epoll\_wait resumed>[{events=EPOLLOUT, data={u32=2818577456, u64=139825478898736}}], 256, -1) = 1

[pid 106407] <... recvfrom resumed>"\4\n\tSUBSCRIBE", 8192, 0, NULL, NULL) = 12

[pid 106627] epoll\_ctl(7, EPOLL\_CTL\_MOD, 9, {events=EPOLLIN, data={u32=2818577456, u64=139825478898736}} <unfinished ...>

[pid 106407] epoll\_wait(7, <unfinished ...>

[pid 106627] <... epoll\_ctl resumed>) = 0

[pid 106405] read(0, exec 3 2 6 7

"exec 3 2 6 7\n", 1024) = 13

[pid 106405] getpid() = 106405

[pid 106405] poll([{fd=8, events=POLLIN}], 1, 0) = 1 ([{fd=8, revents=POLLIN}])

[pid 106405] getpid() = 106405

[pid 106405] read(8, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106405] getpid() = 106405

[pid 106405] poll([{fd=8, events=POLLIN}], 1, 0) = 0 (Timeout)

[pid 106405] getpid() = 106405

[pid 106405] write(6, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106407] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=735224416, u64=94713354035808}}], 256, -1) = 1

[pid 106405] read(0, <unfinished ...>

[pid 106407] getpid() = 106405

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0) = 1 ([{fd=6, revents=POLLIN}])

[pid 106407] getpid() = 106405

[pid 106407] read(6, "\1\0\0\0\0\0\0\0", 8) = 8

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_MOD, 10, {events=EPOLLIN|EPOLLOUT, data={u32=2415922128, u64=140671184800720}}) = 0

[pid 106407] sendto(10, "\0\rexec 3 2 6 7\n", 15, 0, NULL, 0 <unfinished ...>

[pid 106476] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=3623883824, u64=140156996686896}}], 256, -1) = 1

[pid 106407] <... sendto resumed>) = 15

[pid 106476] recvfrom(9, <unfinished ...>

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_MOD, 11, {events=EPOLLIN|EPOLLOUT, data={u32=2415988864, u64=140671184867456}} <unfinished ...>

[pid 106476] <... recvfrom resumed>"\0\rexec 3 2 6 7\n", 8192, 0, NULL, NULL) = 15

[pid 106407] <... epoll\_ctl resumed>) = 0

[pid 106476] getpid( <unfinished ...>

[pid 106407] sendto(11, "\0\rexec 3 2 6 7\n", 15, 0, NULL, 0 <unfinished ...>

[pid 106476] <... getpid resumed>) = 106474

[pid 106627] <... epoll\_wait resumed>[{events=EPOLLIN, data={u32=2818577456, u64=139825478898736}}], 256, -1) = 1

[pid 106476] write(8, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

[pid 106407] <... sendto resumed>) = 15

[pid 106627] recvfrom(9, <unfinished ...>

[pid 106476] <... write resumed>) = 8

[pid 106474] <... poll resumed>) = 1 ([{fd=8, revents=POLLIN}])

[pid 106407] getpid( <unfinished ...>

[pid 106627] <... recvfrom resumed>"\0\rexec 3 2 6 7\n", 8192, 0, NULL, NULL) = 15

[pid 106476] epoll\_wait(7, <unfinished ...>

[pid 106474] getpid( <unfinished ...>

[pid 106627] getpid( <unfinished ...>

[pid 106407] <... getpid resumed>) = 106405

[pid 106627] <... getpid resumed>) = 106625

[pid 106474] <... getpid resumed>) = 106474

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

[pid 106627] write(8, "\1\0\0\0\0\0\0\0", 8 <unfinished ...>

[pid 106474] read(8, <unfinished ...>

[pid 106407] <... poll resumed>) = 0 (Timeout)

[pid 106627] <... write resumed>) = 8

[pid 106474] <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

[pid 106627] epoll\_wait(7, <unfinished ...>

[pid 106625] <... poll resumed>) = 1 ([{fd=8, revents=POLLIN}])

[pid 106626] epoll\_ctl(5, EPOLL\_CTL\_MOD, 8, {events=EPOLLIN, data={u32=2684357488, u64=139825344678768}} <unfinished ...>

[pid 106475] mmap(NULL, 134217728, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_NORESERVE, -1, 0 <unfinished ...>

[pid 106626] <... epoll\_ctl resumed>) = 0

[pid 106475] <... mmap resumed>) = 0x7f78d0000000

[pid 106626] getpid( <unfinished ...>

[pid 106475] munmap(0x7f78d4000000, 67108864 <unfinished ...>

[pid 106626] <... getpid resumed>) = 106625

[pid 106407] <... getpid resumed>) = 106405

[pid 106406] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], <unfinished ...>

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0 <unfinished ...>

[pid 106406] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 106407] <... poll resumed>) = 1 ([{fd=6, revents=POLLIN}])

[pid 106406] madvise(0x7ff095202000, 8368128, MADV\_DONTNEED <unfinished ...>

[pid 106407] getpid() = 106405

[pid 106406] <... madvise resumed>) = 0

[pid 106407] read(6, <unfinished ...>

[pid 106406] exit(0 <unfinished ...>

[pid 106407] <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

[pid 106406] <... exit resumed>) = ?

[pid 106407] epoll\_ctl(7, EPOLL\_CTL\_DEL, 6, 0x56242bd2a264 <unfinished ...>

[pid 106406] +++ exited with 0 +++

[pid 106407] <... epoll\_ctl resumed>) = 0

[pid 106407] getpid() = 106405

[pid 106407] poll([{fd=6, events=POLLIN}], 1, 0) = 0 (Timeout)

[pid 106407] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 106407] madvise(0x7ff094a01000, 8368128, MADV\_DONTNEED) = 0

[pid 106407] exit(0) = ?

[pid 106405] <... futex resumed>) = 0

[pid 106407] +++ exited with 0 +++

close(7) = 0

close(6) = 0

close(5) = 0

close(4) = 0

close(3) = 0

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В результате выполнения данной лабораторной работы я узнал такую важную вещь, как брокер сообщений. Я научился передавать данные между процессами, что очень важно для проектирования систем клиент-сервер. Этот навык очень поможет мне в будущем.