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: М8О-215Б-23

Студент: Тараскаев Д.М.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 01.12.24

Москва, 2024

Постановка задачи

Вариант 43.

Топология узлов: общий список
Тип команды: подсчет суммы n чисел
Тип проверки доступности узлов: ping id

Общий метод и алгоритм решения

В программе использовались следующие системные вызовы:

- execve - запуск исполняемого файла "./client"
- eventfd2 - создание дескриптора для уведомлений между потоками
- socket - создание сокета для сетевой коммуникации
- setsockopt - настройка параметров сокета
- bind - привязка сокета к определенному адресу/порту
- listen - перевод сокета в режим прослушивания для TCP-соединений
- fcntl - настройка неблокирующего режима для файловых дескрипторов
- clone3 - создание потоков для обработки сообщений
- prctl - установка имен для потоков (ZMQbg/Reaper, ZMQbg/IO/0)
- read - чтение данных из сокетов или дескрипторов
- write - запись данных в сокеты или дескрипторы
- sendto - отправка данных через сокет с указанием адресата
- recvmsg - получение данных из сокета с дополнительными метаданными
- close - закрытие файловых дескрипторов и сокетов
- poll - проверка готовности дескрипторов к операциям ввода-вывода
- getpid - получение идентификатора процесса

Код программы

client.cpp

```
#include <signal.h>

#include <atomic>
#include <chrono>
#include <map>
#include <mutex>
#include <thread>

#include "net_func.h"
#include "node.h"
#include "set"

static std::mutex nodes_mutex;

int main() {
    std::set<int> all_nodes;
```

```
std::string prog_path = "./worker";
Node me(-1);
all_nodes.insert(-1);

std::string command;
while (std::cin >> command) {
    if (command == "create") {
        int id_child, id_parent;
        std::cin >> id_child >> id_parent;

        if (all_nodes.find(id_child) != all_nodes.end()) {
            std::cout << "Error: Already exists" << std::endl;
        }
        else if (all_nodes.find(id_parent) == all_nodes.end()) {
            std::cout << "Error: Parent not found" << std::endl;
        }
        else if (id_parent == me.id) {
            std::string ans = me.Create_child(id_child, prog_path);
            std::cout << ans << std::endl;
            all_nodes.insert(id_child);
        }
        else {
            std::string str = "create " + std::to_string(id_child);
            std::string ans = me.Send(str, id_parent);
            std::cout << ans << std::endl;
            all_nodes.insert(id_child);
        }
    }
    else if (command == "ping") {
        int id_child;
        std::cin >> id_child;
        if (all_nodes.find(id_child) == all_nodes.end()) {
            std::cout << "Error: Not found" << std::endl;
        }
        else if (me.children.find(id_child) != me.children.end()) {
            std::string ans = me.Ping_child(id_child);
            std::cout << ans << std::endl;
        }
        else {
            std::string str = "ping " + std::to_string(id_child);
            std::string ans = me.Send(str, id_child);
            if (ans == "Error: not find") {
                ans = "Ok: 0";
            }
            std::cout << ans << std::endl;
        }
    }
    else if (command == "exec") {
        int id, n;
```

```
            std::cin >> id >> n;

            // Формируем сообщение с количеством чисел и самими числами
            std::string msg = "exec " + std::to_string(n);
            for (int i = 0; i < n; i++) {
                int num;
                std::cin >> num;
                msg += " " + std::to_string(num);
            }

            if (all_nodes.find(id) == all_nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
            }
            else {
                std::string ping_result =
                    me.Send("ping " + std::to_string(id), id);
                if (ping_result == "Ok: 0" ||
                    ping_result == "Error: not find") {
                    std::cout << "Error: Node is unavailable" << std::endl;
                    all_nodes.erase(id);
                }
                else {
                    std::string ans = me.Send(msg, id);
                    std::cout << ans << std::endl;
                }
            }
        }
    }

    me.Remove();
    return 0;
}
```

worker.cpp

```
#include <signal.h>

#include <chrono>
#include <fstream>
#include <vector>

#include "net_func.h"
#include "node.h"

int my_id = 0;

int main(int argc, char **argv) {
    if (argc != 3) {
```

```
    return -1;
}

Node me(atoi(argv[1]), atoi(argv[2]));
my_id = me.id;
std::string prog_path = "./worker";
while (1) {
    if (getppid() == 1) {  // Если родительский процесс мертв, завершаем узел
        std::cout << "[WORKER] Parent process died. Exiting...\n";
        break;
    }
    std::string message;
    std::string command = " ";
    message = my_net::reseave(&(me.parent));
    std::istringstream request(message);
    request >> command;

    if (command == "create") {
        int id_child;
        request >> id_child;
        std::string ans = me.Create_child(id_child, prog_path);
        my_net::send_message(&me.parent, ans);
    }
    else if (command == "pid") {
        std::string ans = me.Pid();
        my_net::send_message(&me.parent, ans);
    }
    else if (command == "ping") {
        int id_child;
        request >> id_child;
        std::string ans = me.Ping_child(id_child);
        my_net::send_message(&me.parent, ans);
    }
    else if (command == "send") {
        int id;
        request >> id;
        std::string str;
        getline(request, str);
        str.erase(0, 1);
        std::string ans;
        ans = me.Send(str, id);
        my_net::send_message(&me.parent, ans);
    }
    else if (command == "exec") {
        int n;
        request >> n;
        int sum = 0;

        // Считываем и суммируем n чисел
```

```
            for (int i = 0; i < n; i++) {
                int num;
                request >> num;
                sum += num;
            }

            std::string ans =
                "Ok:" + std::to_string(me.id) + ":" + std::to_string(sum);
            my_net::send_message(&me.parent, ans);
        }
    }

    sleep(1);
    return 0;
}
```

net_func.h

```
#pragma once

#include <iostream>
#include <sstream>
#include <string>
#include <zmq.hpp>

namespace my_net {

#define MY_PORT 59000
#define MY_IP "tcp://localhost:"

int bind(zmq::socket_t *socket, int id) {
    int port = MY_PORT + id;
    while (true) {
        std::string adress = MY_IP + std::to_string(port);
        try {
            socket->bind(adress);
            break;
        } catch (...) {
            port++;
        }
    }
    return port;
}

void connect(zmq::socket_t *socket, int port) {
    std::string adress = MY_IP + std::to_string(port);
    socket->connect(adress);
```

```
}

void unbind(zmq::socket_t *socket, int port) {
    std::string adress = MY_IP + std::to_string(port);
    socket->unbind(adress);
}

void disconnect(zmq::socket_t *socket, int port) {
    std::string adress = MY_IP + std::to_string(port);
    socket->disconnect(adress);
}

void send_message(zmq::socket_t *socket, const std::string msg) {
    zmq::message_t message(msg.size());
    memcpy(message.data(), msg.c_str(), msg.size());
    try {
        socket->send(message);
    } catch (...) {
    }
}

std::string reseave(zmq::socket_t *socket) {
    zmq::message_t message;
    bool success = true;
    try {
        socket->recv(&message, 0);
    } catch (...) {
        success = false;
    }
    if (!success || message.size() == 0) {
        throw -1;
    }
    std::string str(static_cast<char *>(message.data()), message.size());
    return str;
}

}  // namespace my_net
urn a + b;
}
```

node.h

```
#include <iostream>
#include <sstream>
#include <unordered_map>

#include "net_func.h"
#include "unistd.h"

class Node {
```

```
private:
 zmq::context_t context;

public:
 std::unordered_map<int, zmq::socket_t *> children;
 std::unordered_map<int, int> children_port;
 zmq::socket_t parent;
 int parent_port;
 int id;

 Node(int _id, int _parent_port = -1)
     : parent(context, ZMQ_REP), parent_port(_parent_port), id(_id) {
     if (_id != -1) {
         my_net::connect(&parent, _parent_port);
     }
 }

 std::string Ping_child(int _id) {
     std::string ans = "Ok: 0";
     if (_id == id) {
         ans = "Ok: 1";
         return ans;
     } else if (children.find(_id) != children.end()) {
         std::string msg = "ping " + std::to_string(_id);
         my_net::send_message(children[_id], msg);
         try {
             msg = my_net::reseave(children[_id]);
             if (msg == "Ok: 1") ans = msg;
         }
         catch (int) {
         }
         return ans;
     }
     else {
         return ans;
     }
 }

 std::string Create_child(int child_id, std::string program_path) {
     std::string program_name =
         program_path.substr(program_path.find_last_of("/") + 1);
     children[child_id] = new zmq::socket_t(context, ZMQ_REQ);

     int new_port = my_net::bind(children[child_id], child_id);
     children_port[child_id] = new_port;
     int pid = fork();

     if (pid == 0) {
         execl(program_path.c_str(), program_name.c_str(),
```

```
                std::to_string(child_id).c_str(),
                std::to_string(new_port).c_str(), (char *)NULL);
    } else {
        std::string child_pid;
        try {
            children[child_id]->setsockopt(ZMQ_SNDTIMEO, 3000);
            my_net::send_message(children[child_id], "pid");
            child_pid = my_net::reseave(children[child_id]);
        }
        catch (int) {
            child_pid = "Error: can't connect to child";
        }
        return "Ok: " + child_pid;
    }
}

std::string Pid() { return std::to_string(getpid()); }

std::string Send(std::string str, int _id) {
    if (_id == id) {
        return "Ok: 1";
    }
    if (children.find(_id) != children.end()) {
        if (Ping_child(_id) == "Ok: 1") {
            my_net::send_message(children[_id], str);
            try {
                return my_net::reseave(children[_id]);
            }
            catch (int) {
                return "Error: not find";
            }
        }
    }
    for (auto &child : children) {
        std::string msg = "send " + std::to_string(_id) + " " + str;
        my_net::send_message(children[child.first], msg);
        try {
            std::string response = my_net::reseave(children[child.first]);
            if (response != "Error: not find") {
                return response;
            }
        }
        catch (int) {
        }
    }
    return "Error: not find";
}

std::string Remove() {
```

```
        std::string ans;
        if (children.size() > 0) {
            for (auto &child : children) {
                if (Ping_child(child.first) == "Ok: 1") {
                    std::string msg = "remove";
                    my_net::send_message(children[child.first], msg);
                    try {
                        msg = my_net::reseave(children[child.first]);
                        if (ans.size() > 0) ans = ans + " " + msg;
                        else ans = msg;
                    }
                    catch (int) { }
                }
                my_net::unbind(children[child.first],
                               children_port[child.first]);
                children[child.first]->close();
            }
            children.clear();
            children_port.clear();
        }
        return ans;
    }
};
```

Протокол работы программы

Тестирование

```
$ ./client
create 1 -1
Ok: 13019
create 2 1
Ok: 13026
create 3 2
Ok: 13029
exec 3 2 1 3
Ok:3:4
ping 2
Ok: 1
(В другом терминале $ kill -9 13026)
ping 2
Ok: 0
exit
```

strace:

```
$ strace ./client
14163 eventfd2(0, EFD_CLOEXEC)          = 3
```

```
     14163 fcntl(3, F_GETFL)                 = 0x2 (flags O_RDWR)

     14163 fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK) = 0
     14163 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 10

     14163 setsockopt(10, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0

     14163 bind(10, {sa_family=AF_INET, sin_port=htons(59001),
sin_addr=inet_addr("127.0.0.1")}, 16) = 0

     14163 listen(10, 100)                   = 0
     14163
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_S
ETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...

     14164 prctl(PR_SET_NAME, "ZMQbg/Reaper")  = 0

     ...

     14165 prctl(PR_SET_NAME, "ZMQbg/IO/0")  = 0

     14163 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
...

     14185 execve("./worker", ["worker", "1", "59001"], 0x7ffcb02eb0c8 /* 51 vars */)
```

Вывод

Программа успешно демонстрирует работу с очередью сообщений.