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»

Управлении серверами сообщений. Применение отложенных вычислений. Интеграция программных систем друг с другом

Студент: Преподаватель:

Группа:

Вариант:

Дата: Оценка: Подпись:

Селивёрстов Д. С.

Миронов Е.С.

М8О-201Б-21

13

Москва, 2023

1 Постановка задачи

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вы-числительный». Необходимо объединить данные узлы в соответствии с той тополо-гией, 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 убий-стве («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 работоспособ-ность.

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

Топология – дерево общего вида, проверка – доступности ping id, вычислительная программа – поиск подстоки в строке.

2 Общие сведения о программе

Программа написанна на языке Си в Unix подобной операционной системе на базе ядра Linux. В программе используется очередь сообщений ZeroMQ.

Программа поддерживает следующие команды:

1. create [id] [parent] – создать узел с id [id], родителем которого является узел с id [parent].

2. remove [id] – удаляет узел с данным id.

3. exec [id] [string] [pattern] – запускает на узле [id] поиск подстроки [pattern] в строке [string].

4. ping [id] – проверка доступности узда [id].

Выход из программы происзодит при окончании ввода, то есть при нажатии CTRL+D.

3 Общий метод и алгоритм решения

В программе используется тип соединения Request-Response. Узлы прередают друг другу информацию при помощи очереди сообщений.

1

Все функции описанны либо для самого целевого узла либо его родителя. А для доставки сообщений создан метод Send(). Этот методо проверяет на доступность всех детей, если они доступны то отправляет им сообщение. В случае если детей нет, то он возвращает родителю ошибку об не нахождении узла. Когда родитель получает ответы от всех детей он отправляет на уровень выше либо единственный отличающийся от ошибки поиска, либо саму ошибку поиска. И так пока ответ не достигнет отправителя.

Если команда обращается к несуществующему узлу, то мы сразу возвращаем ошибку. Для этого у нас есть множество созданных узлов.

При создании мы просо форкаем родитель и передаем ребенку данные для связи с ним.

При удалении же мы передаем все детям также сигнал об удалении и рекурсив-но формируем список всех своих потомков. В который по итогу добавляем себя и отправляем родителю. По итогу клиент удаляет все элименты этого списка из мно-жества созданных узлов.

Для проверки доступности мы посылаем ребенку сообщение, и если за 3 секунды не получаем ответа, то считаем узел не доступным.

Поиск подстроки в строке происходит с помощью наивного алгорима поиска из стан-дарной библиотеки.

4 Листинг программы

net\_func.hpp

1 #pragma once 2

3 #include <iostream> 4 #include <zmq.hpp> 5 #include <sstream> 6 #include <string>

7

8 namespace my\_net{

9 #define MY\_PORT 4040

10 #define MY\_IP "tcp://127.0.0.1:" 11

12 int bind(zmq::socket\_t \*socket, int id){ 13 int port = MY\_PORT + id;

14 while(true){

15 std::string adress = MY\_IP + std::to\_string(port); 16 try{

17 socket->bind(adress); 18 break;

19 } catch(...){

2

20 port++; 21 }

22 }

23 return port; 24 }

25

26 void connect(zmq::socket\_t \*socket, int port){

27 std::string adress = MY\_IP + std::to\_string(port); 28 socket->connect(adress);

29 } 30

31 void unbind(zmq::socket\_t \*socket, int port) {

32 std::string adress = MY\_IP + std::to\_string(port); 33 socket->unbind(adress);

34 } 35

36 void disconnect(zmq::socket\_t \*socket, int port) {

37 std::string adress = MY\_IP + std::to\_string(port); 38 socket->disconnect(adress);

39 } 40

41 void send\_message(zmq::socket\_t \*socket, const std::string msg) { 42 zmq::message\_t message(msg.size());

43 memcpy(message.data(), msg.c\_str(), msg.size()); 44 try{

45 socket->send(message); 46 }catch(...){}

47 } 48

49 std::string reseave(zmq::socket\_t \*socket){ 50 zmq::message\_t message;

51 bool success = true; 52 try{

53 socket->recv(&message,0); 54 }catch(...){

55 success = false; 56 }

57 if(!success || message.size() == 0){; 58 throw -1;

59 }

60 std::string str(static\_cast<char\*>(message.data()), message.size()); 61 return str;

62 } 63 }

node.hpp

1 #include <iostream>

2 #include "net\_func.hpp" 3 #include <sstream>

3

4 #include <unordered\_map> 5 #include "unistd.h"

6 7

8 class Node{ 9 private:

10 zmq::context\_t context; 11 public:

12 std::unordered\_map<int,zmq::socket\_t\*> children; 13 std::unordered\_map<int,int> children\_port;

14 zmq::socket\_t parent; 15 int parent\_port;

16

17 int id;

18 Node(int \_id , int \_parent\_port = -1): id(\_id),

19 parent(context,ZMQ\_REP), 20 parent\_port(\_parent\_port){ 21 if(\_id != -1){

22 my\_net::connect(&parent,\_parent\_port); 23 }

24 } 25

26 std::string Ping\_child(int \_id){ 27 std::string ans = "Ok: 0";

28 ans = "Ok: 0"; 29 if(\_id == id){

30 ans = "Ok: 1"; 31 return ans;

32 } else if(children.find(\_id) != children.end()){

33 std::string msg = "ping " + std::to\_string(\_id); 34 my\_net::send\_message(children[\_id],msg);

35 try{

36 msg = my\_net::reseave(children[\_id]); 37 if(msg == "Ok: 1")

38 ans = msg; 39 } catch(int){}

40 return ans; 41 }else{

42 return ans; 43 }

44 } 45

46 std::string Create\_child(int child\_id,std::string program\_path){

47 std::string program\_name = program\_path.substr(program\_path.find\_last\_of("/") + 1);

48 children[child\_id] = new zmq::socket\_t(context,ZMQ\_REQ); 49

50 int new\_port = my\_net::bind(children[child\_id],child\_id); 51 children\_port[child\_id] = new\_port;

4

52 int pid = fork(); 53

54 if(pid == 0){

55 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);

56 }else{

57 std::string child\_pid; 58 try{

59 children[child\_id]->setsockopt(ZMQ\_SNDTIMEO,3000); 60 my\_net::send\_message(children[child\_id],"pid");

61 child\_pid = my\_net::reseave(children[child\_id]); 62 } catch(int){

63 child\_pid = "Error: can't connect to child"; 64 }

65 return "Ok: " + child\_pid; 66 }

67 } 68

69 std::string Pid(){

70 return std::to\_string(getpid()); 71 }

72

73 std::string Send(std::string str,int \_id){ 74 if(children.size() == 0){

75 return "Error: now find";

76 }else if(children.find(\_id) != children.end()){ 77 if(Ping\_child(\_id) == "Ok: 1"){

78 my\_net::send\_message(children[\_id],str); 79 std::string ans;

80 try{

81 ans = my\_net::reseave(children[\_id]); 82 } catch(int){

83 ans = "Error: now find"; 84 }

85 return ans; 86 }

87 }else{

88 std::string ans = "Error: not find"; 89 for(auto& child: children ){

90 if(Ping\_child(child.first) == "Ok: 1"){

91 std::string msg = "send " + std::to\_string(\_id) + " " + str; 92 my\_net::send\_message(children[child.first],msg);

93 try{

94 msg = my\_net::reseave(children[child.first]); 95 } catch(int){

96 msg = "Error: not find"; 97 }

98 if(msg != "Error: not find"){ 99 ans = msg;

5

100 } 101 }

102 }

103 return ans; 104 }

105 } 106

107 std::string Remove(){ 108 std::string ans;

109 if(children.size() > 0){

110 for(auto& child: children ){

111 if(Ping\_child(child.first) == "Ok: 1"){ 112 std::string msg = "remove";

113 my\_net::send\_message(children[child.first],msg); 114 try{

115 msg = my\_net::reseave(children[child.first]); 116 if(ans.size() > 0)

117 ans = ans + " " + msg; 118 else

119 ans = msg; 120 } catch(int){} 121 }

122 my\_net::unbind(children[child.first], children\_port[child.first]); 123 children[child.first]->close();

124 }

125 children.clear();

126 children\_port.clear(); 127 }

128 return ans; 129 }

130 };

worker.cpp

1 #include "node.hpp"

2 #include "net\_func.hpp" 3 #include <fstream>

4 #include <vector>

5 #include <signal.h> 6

7 int my\_id = 0; 8

9 void Log(std::string str){

10 std::string f = std::to\_string(my\_id) + ".txt"; 11 std::ofstream fout(f,std::ios\_base::app);

12 fout << str; 13 fout.close(); 14 }

15

16 int main(int argc, char \*\*argv){

6

17 if(argc != 3){ 18 return -1; 19 }

20

21 Node me(atoi(argv[1]),atoi(argv[2])); 22 std::string prog\_path = "./worker"; 23 while(1){

24 std::string message;

25 std::string command = " ";

26 message = my\_net::reseave(&(me.parent)); 27 std::istringstream request(message);

28 request >> command; 29

30

31 if(command == "create"){

32 int id\_child, id\_parent; 33 request >> id\_child;

34 std::string ans = me.Create\_child(id\_child, prog\_path); 35 my\_net::send\_message(&me.parent,ans);

36 } else if(command == "pid"){

37 std::string ans = me.Pid();

38 my\_net::send\_message(&me.parent,ans); 39 } else if(command == "ping"){

40 int id\_child;

41 request >> id\_child;

42 std::string ans = me.Ping\_child(id\_child); 43 my\_net::send\_message(&me.parent,ans);

44 } else if(command == "send"){ 45 int id;

46 request >> id; 47 std::string str;

48 getline(request, str); 49 str.erase(0,1);

50 std::string ans;

51 ans = me.Send(str,id);

52 my\_net::send\_message(&me.parent,ans); 53 } else if(command == "exec"){

54 std::string str;

55 std::string pattern;

56 request >> str >> pattern; 57 std::vector<int> answers; 58 int start = 0;

59 while(str.find(pattern,start) != -1){ 60 start = str.find(pattern,start); 61 answers.push\_back(start);

62 start++; 63 }

64 std::string to\_send;

65 if(answers.size() == 0){

7

66 to\_send = "-1"; 67 }else{

68 to\_send = std::to\_string(answers[0]); 69 for(int i = 1; i < answers.size();++i){

70 to\_send = to\_send + ";" + std::to\_string(answers[i]); 71 }

72 }

73 to\_send = "Ok:" + std::to\_string(me.id) + ":" + to\_send; 74 my\_net::send\_message(&me.parent,to\_send);

75 } else if(command == "remove"){

76 std::string ans = me.Remove();

77 ans = std::to\_string(me.id) + " " + ans; 78 my\_net::send\_message(&me.parent, ans);

79 my\_net::disconnect(&me.parent, me.parent\_port); 80 me.parent.close();

81 break; 82 }

83 }

84 sleep(1); 85 return 0; 86 }

client.cpp

1 #include "node.hpp"

2 #include "net\_func.hpp" 3 #include "set"

4 #include <signal.h> 5

6 7

8 int main(){

9 std::set<int> all\_nodes;

10 //std::set<int> not\_availvable;

11 std::string prog\_path = "./worker"; 12 Node me(-1);

13 all\_nodes.insert(-1); 14 std::string command;

15 while(std::cin >> command){ 16 if(command == "create"){

17 int id\_child, id\_parent;

18 std::cin >> id\_child >> id\_parent;

19 if(all\_nodes.find(id\_child) != all\_nodes.end()){

20 std::cout << "Error: Already exists" << std::endl;

21 } else if(all\_nodes.find(id\_parent) == all\_nodes.end()){ 22 std::cout << "Error: Parent not found" << std::endl; 23 }else if(id\_parent == me.id){

24 std::string ans = me.Create\_child(id\_child, prog\_path); 25 std::cout << ans << std::endl;

26 all\_nodes.insert(id\_child);

8

27 } else{

28 std::string str = "create " + std::to\_string(id\_child); 29 std::string ans = me.Send(str, id\_parent);

30 std::cout << ans << std::endl; 31 all\_nodes.insert(id\_child);

32 }

33 } else if(command == "ping"){ 34 int id\_child;

35 std::cin >> id\_child;

36 if(all\_nodes.find(id\_child) == all\_nodes.end()){ 37 std::cout << "Error: Not found" << std::endl;

38 }else if(me.children.find(id\_child) != me.children.end()){ 39 std::string ans = me.Ping\_child(id\_child);

40 std::cout << ans << std::endl; 41 }else{

42 std::string str = "ping " + std::to\_string(id\_child); 43 std::string ans = me.Send(str, id\_child);

44 if(ans == "Error: not find"){ 45 ans = "Ok: 0";

46 }

47 std::cout << ans << std::endl; 48 }

49 }else if(command == "exec"){ 50 int id;

51 std::string str;

52 std::string pattern;

53 std::cin >> id >> str >> pattern;

54 std::string msg = "exec " + str + " " + pattern; 55 if(all\_nodes.find(id) == all\_nodes.end()){

56 std::cout << "Error: Not found" << std::endl; 57 }else{

58 std::string ans = me.Send(msg,id); 59 std::cout << ans << std::endl;

60 } 61

62 }else if(command == "remove"){ 63 int id;

64 std::cin >> id;

65 std::string msg = "remove";

66 if(all\_nodes.find(id) == all\_nodes.end()){

67 std::cout << "Error: Not found" << std::endl; 68 }else{

69 std::string ans = me.Send(msg,id); 70 if(ans != "Error: not find"){

71 std::istringstream ids(ans); 72 int tmp;

73 while(ids >> tmp){

74 all\_nodes.erase(tmp); 75 }

9

76 ans = "Ok";

77 if(me.children.find(id) != me.children.end()){

78 my\_net::unbind(me.children[id],me.children\_port[id]); 79 me.children[id]->close();

80 me.children.erase(id);

81 me.children\_port.erase(id); 82 }

83 }

84 std::cout << ans << std::endl; 85 }

86 } 87 }

88 me.Remove(); 89 return 0;

90 }

10

5 Демонстрация работы программы

dmitri@dmitri-laptop:~/ClionProjects/os\_labs/lab6-8$ ./client create 2 -1

Ok: 49589 create 3 2 Ok: 49593 create 4 3 Ok: 49597 create 5 3 Ok: 49601 ping 5

Ok: 1 exec 3

abracadabra abra Ok:3:0;7 remove 5

Ok ping 5

Error: Not found remove 3

ping 3 Ok

Error: Not found remove 4

Error: Not found

11

6 Вывод

Наряду с каналами и отображаемыми файлами для передачи данных применяются очереди сообщений.

Данная лабораторная работа была направлена на изучение технологий очередей со-общений и сокетов. Я ознакомился и изучил их, реализовав сеть с заданной топо-логей. При создании я использовал библиотеку для передачи сообщений ZeroMQ. ZeroMQ предоставляет простой интерфейс для передачи сообщений и составления топологий разных типов.

Полученные навыки можно применить во многих сферах, так как большинство про-грамм сейчас взаимодействуют по сети. При этом ZeroMQ можно применить и для внутреннего межпроцессорного взаимодействия.

12