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: В. А. Амурский
Преподаватель: Е. С. Миронов
Группа: М8О-201Б-21
Вариант: 23
Дата:
Оценка:
Подпись:

Москва, 2023

1 Постановка задачи

Необходимо создать распределенную систему для асинхронной обработки запросов, включающую два типа узлов: управляющие и вычислительные. Соединение между узлами должно быть осуществлено через технологию очередей сообщений в соответствии с топологией, заданной вариантом. Также необходимо реализовать проверку доступности узлов согласно варианту. Если вычислительный узел убит с помощью команды "kill -9 система должна сохранять работоспособность, позволяя дочерним узлам быть недоступными, но обеспечивая работу родительских узлов.

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.

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 рекурсивно формируем список всех своих потомков. В который по итогу добавляем себя и отправляем родителю. По итогу клиент удаляет все элименты этого списка из множества 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 стандарной 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(...){
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>
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;
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;
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){
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){
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]);
```

```
                }
            }
            to_send = "Ok:" + std::to_string(me.id) + ":" + to_send;
            my_net::send_message(&me.parent,to_send);
        } else if(command == "remove"){
            std::string ans = me.Remove();
            ans = std::to_string(me.id) + " " + ans;
            my_net::send_message(&me.parent, ans);
            my_net::disconnect(&me.parent, me.parent_port);
            me.parent.close();
            break;
        }
    }
    sleep(1);
    return 0;
}
```

client.cpp

```
#include "node.hpp"
#include "net_func.hpp"
#include "set"
#include <signal.h>



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

```
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                      }
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);
```

```
                    me.children_port.erase(id);
                }
            }
            std::cout << ans << std::endl;
        }
    }
    me.Remove();
    return 0;
}
```

5 Демонстрация работы программы

```
botashev@botashev-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
```

6 Вывод

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

Данная лабораторная работа была направлена на изучение технологий очередей сообщений и сокетов. Я ознакомился и изучил их, реализовав сеть с заданной топологей. 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 программ сейчас взаимодействуют по сети. При этом ZeroMQ можно применить и для внутреннего межпроцессорного взаимодействия.