**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 6-8**

Тема: Очереди сообщений

Студент: Калугин К. А.

Группа: М8О-207Б-19

Преподаватель: Миронов Е.С.

Дата:

Оценка:

Москва, 2021

**Постановка задачи:** 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.

**Алгоритм решения задачи.**

Лабораторная работа была выполнена в упрощенном виде. Управляющий узел создает вычислительный узел и связывается с ним при помощи технологии ZeroMQ. Доступно выполнение команды exec, реализующей целочисленный словарь, основанный на map.

**Листинг программы**

#include <iostream>

#include <string>

#include "zmq.h"

#include <windows.h>

#include <vector>

using namespace std;

struct DataForConnection {

    void \* context;

    void \* requester;

    int childID;

    void Create (int id) {

        childID = id;

        context = zmq\_ctx\_new ();

        requester = zmq\_socket (context, ZMQ\_PAIR);

        string s = "tcp://localhost:";

        s += to\_string (id);

        zmq\_connect (requester, s.c\_str ());

    }

    void Close () {

        zmq\_close (requester);

        zmq\_ctx\_destroy (context);

    }

};

vector <DataForConnection> children;

int CreateNode (int nID) {

    STARTUPINFO si;

    ZeroMemory (&si, sizeof (STARTUPINFO));

    si.cb = sizeof (STARTUPINFO);

    PROCESS\_INFORMATION pi;

    ZeroMemory (&pi, sizeof (PROCESS\_INFORMATION));

    string strCommand = "lr6-8calc.exe ";

    strCommand += to\_string (nID);

    char command [20];

    for (int i = 0; i < strCommand.size (); i ++) {

        command [i] = strCommand [i];

    }

    command [strCommand.size ()] = '\0';

    bool creationSuccessful = false;

    creationSuccessful = CreateProcess (

        NULL, // The name of the module to be executed. No module name. Uses command line.

        command, // Text in command line.

        NULL, // Process gets a default security descriptor.

        NULL, // Thread gets a default security descriptor.

        TRUE, // Inheritable handles are inherited by the new process.

        0, // Process creation flags.

        NULL, // New process uses the environment of the calling process.

        NULL, // New process will have the same current directory as the calling process.

        &si,

        &pi

    );

    if (!creationSuccessful) {

        return -1;

    }

    DataForConnection dfc;

    dfc.Create (nID);

    children.push\_back (dfc);

    return pi.dwProcessId;

}

void DestroyNode (int number) {

    zmq\_send (children [number].requester, "e", 1, 0);

    children [number].Close ();

}

struct Data {

    int nodeID;

    int Create (int idx) {

        nodeID = idx;

        return 0;

    }

    void Destroy () {}

};

struct Node {

    Data data;

    Node \* brother;

    Node \* son;

};

Node \* Search (Node \* curr, int idx, vector <long int> path, vector <long int> & retPath) {

    Node \* res = nullptr;

    while (curr != nullptr) {

        if (curr->data.nodeID == idx) {

            retPath = path;

            return curr;

        }

        path.push\_back (0);

        res = Search (curr->son, idx, path, retPath);

        if (res != nullptr) {

            return res;

        }

        path.pop\_back ();

        path [path.size () - 1] ++;

        curr = curr->brother;

    }

    return nullptr;

}

bool SearchForParent (Node \*\* prev, Node \* curr, int idx, int \* flag) {

    bool res = false;

    while (curr != nullptr) {

        if (curr->data.nodeID == idx) {

            return true;

        }

        \*prev = curr;

        \*flag = 1;

        res = SearchForParent (prev, curr->son, idx, flag);

        if (res) {

            return true;

        }

        \*prev = curr;

        \*flag = 2;

        curr = curr->brother;

    }

    return false;

}

int Insert (Node \* parent, int idx) {

    if (parent == nullptr) {

        return -2;

    }

    int res = 0;

    Node \* brother = parent->son;

    parent->son = new Node;

    parent->son->brother = brother;

    parent->son->son = nullptr;

    res = parent->son->data.Create (idx);

    if (res != 0) {

        delete parent->son;

        parent->son = brother;

    }

    return res;

}

void Delete (Node \*\* curr) {

    \*curr = nullptr;

}

void Print (Node \* curr, int spaces) {

    while (curr != nullptr) {

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

            cout << ' ';

        }

        cout << curr->data.nodeID << endl;

        Print (curr->son, spaces + 3);

        curr = curr->brother;

    }

}

void Clean (Node \* curr) {

    if (curr == nullptr) {

        return;

    }

    Clean (curr->brother);

    Clean (curr->son);

    curr->data.Destroy ();

    delete curr;

}

class Tree {

    private:

    Node \* root;

    public:

    void Init () {

        root = new Node;

        root->data.nodeID = -1;

        root->son = nullptr;

        root->brother = nullptr;

    }

    void InsertInTree (int idx, int parentID) {

        vector <long int> path;

        path.push\_back (0);

        Node \* check = Search (root, idx, path, path);

        if (check != nullptr) {

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

            return;

        }

        if (parentID == -1) {

            int procID = CreateNode (idx);

            int res = Insert (root, idx);

            if (res != 0) {

                cout << "   ERROR: " << '<' << res << '>' << endl;

            }

            else {

                cout << "Ok: " << procID << endl;

            }

            return;

        }

        path.clear ();

        path.push\_back (0);

        Node \* parent = Search (root, parentID, path, path);

        int res = Insert (parent, idx);

        if (res != 0) {

            cout << "Error: Parent not found" << endl;

            return;

        }

        string str = "n";

        str += to\_string (path.size () - 2);

        str += ' ';

        for (int i = 2; i < path.size (); i ++) {

            str += to\_string (path [i]);

            str += ' ';

        }

        str += to\_string (idx);

        str += ' ';

        char bufferToSend [1001] = {'\0'};

        for (int i = 0; i < str.size (); i ++) {

            bufferToSend [i] = str [i];

        }

        zmq\_send (children [path [1]].requester, bufferToSend, 1000, 0);

    }

    int DeleteFromTree (int idx) {

        if (idx < 0) {

            return -4;

        }

        Node \* parent;

        int flag;

        bool check = SearchForParent (&parent, root, idx, &flag);

        if (!check) {

            return -3;

        }

        if (flag == 1) {

            Delete (&parent->son);

        }

        else {

            Delete (&parent->brother);

        }

        return 0;

    }

    void ExecCommand (int id, string & word, string & key) {

        vector <long int> path;

        path.push\_back (0);

        Node \* check = Search (root, id, path, path);

        string str = "c";

        str += to\_string (path.size () - 2);

        str += ' ';

        for (int i = 2; i < path.size (); i ++) {

            str += to\_string (path [i]);

            str += ' ';

        }

        str += word;

        str += ' ';

        str += key;

        str += ' ';

        char bufferToSend [1001] = {'\0'};

        for (int i = 0; i < str.size (); i ++) {

            bufferToSend [i] = str [i];

        }

        zmq\_send (children [path [1]].requester, bufferToSend, 1000, 0);

        // send data for calc

    }

    void PrintTree () {

        Print (root, 1);

    }

    void CleanTree () {

        Clean (root->son);

        delete root;

    }

};

int main () {

    Tree tree;

    tree.Init ();

    string option;

    while (1) {

        cin >> option;

        if (option == "create") {

            int id, parentID;

            cin >> id >> parentID;

            tree.InsertInTree (id, parentID);

        }

        else if (option == "remove") {

            int id, res;

            cin >> id;

            res = tree.DeleteFromTree (id);

            if (res != 0) {

                cout << "   ERROR: " << '<' << res << '>' << endl;

            }

        }

        else if (option == "exec") {

            int id;

            cin >> id;

            string key = "", word = "";

            cin >> word;

            char c = getchar ();

            if (c == ' ') {

                cin >> key;

            }

            else {

                key = "n";

            }

            tree.ExecCommand (id, word, key);

        }

        else if (option == "print") {

            tree.PrintTree ();

        }

        else {

            break;

        }

    }

    tree.CleanTree ();

    cout << " Program finished successfully." << endl;

    return 0;

}

/\*

g++ .\lab6-ctrl-v3.cpp -o control.exe -L. -llibzmq-v142-mt-gd-4\_3\_5

\*/

#include <iostream>

#include <windows.h>

#include "zmq.h"

#include <string>

#include <vector>

#include <map>

using namespace std;

struct DataForConnection {

    void \* context;

    void \* requester;

    int childID;

    void Create (int id) {

        childID = id;

        context = zmq\_ctx\_new ();

        requester = zmq\_socket (context, ZMQ\_PAIR);

        string s = "tcp://localhost:";

        s += to\_string (id);

        zmq\_connect (requester, s.c\_str ());

    }

    void Close () {

        zmq\_close (requester);

        zmq\_ctx\_destroy (context);

    }

};

vector <DataForConnection> children;

int CreateNode (long int nID) {

    STARTUPINFO si;

    ZeroMemory (&si, sizeof (STARTUPINFO));

    si.cb = sizeof (STARTUPINFO);

    PROCESS\_INFORMATION pi;

    ZeroMemory (&pi, sizeof (PROCESS\_INFORMATION));

    string strCommand = "lr6-8calc.exe ";

    strCommand += to\_string (nID);

    char command [20];

    for (int i = 0; i < strCommand.size (); i ++) {

        command [i] = strCommand [i];

    }

    command [strCommand.size ()] = '\0';

    bool creationSuccessful = false;

    creationSuccessful = CreateProcess (

        NULL, // The name of the module to be executed. No module name. Uses command line.

        command, // Text in command line.

        NULL, // Process gets a default security descriptor.

        NULL, // Thread gets a default security descriptor.

        TRUE, // Inheritable handles are inherited by the new process.

        0, // Process creation flags.

        NULL, // New process uses the environment of the calling process.

        NULL, // New process will have the same current directory as the calling process.

        &si,

        &pi

    );

    if (!creationSuccessful) {

        return -1;

    }

    DataForConnection dfc;

    dfc.Create (nID);

    children.push\_back (dfc);

    return pi.dwProcessId;

}

void DestroyNode (int number) {

    zmq\_send (children [number].requester, "e", 1, 0);

    children [number].Close ();

}

long int thisNodeID;

void \* context;

void \* responder;

int main (int argc, char \*\* argv) {

    thisNodeID = strtol (argv [1], NULL, 10);

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

    void \* responder = zmq\_socket (context, ZMQ\_PAIR);

    string s = "tcp://\*:";

    s += argv [1];

    zmq\_bind (responder, s.c\_str ());

    map <string, long int> dictionary;

    while (1) {

        int iter = 1;

        char buffer [1001] = {'\0'};

        zmq\_recv (responder, buffer, 1000, 0);

        if (buffer [0] == 'e') {

            break;

        }

        else if (buffer [0] == 'n') {

            char cNumber1 [11];

            int n = 0;

            while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                cNumber1 [n] = buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            long int messageSize = strtol (cNumber1, NULL, 10);

            if (messageSize == 0) {

                char cNumber4 [11];

                while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                    cNumber4 [n] = buffer [iter];

                    n ++;

                    iter ++;

                }

                long int nID = strtol (cNumber4, NULL, 10);

                int procID = CreateNode (nID);

                cout << "Ok: " << procID << endl;

                continue;

            }

            char cNumber2 [11];

            while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                cNumber2 [n] = buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            long int currChild = strtol (cNumber2, NULL, 10);

            vector <long int> path;

            for (long int i = 0; i < messageSize - 1; i ++) {

                char cNumber3 [11];

                while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                    cNumber3 [n] = buffer [iter];

                    n ++;

                    iter ++;

                }

                n = 0;

                iter ++;

                path.push\_back (strtol (cNumber3, NULL, 10));

            }

            char cNumber4 [11];

            while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                cNumber2 [n] = buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            long int nID = strtol (cNumber4, NULL, 10);

            string str = "n";

            str += to\_string (path.size ());

            str += ' ';

            for (int i = 0; i < path.size (); i ++) {

                str += to\_string (path [i]);

                str += ' ';

            }

            str += to\_string (nID);

            str += ' ';

            char bufferToSend [1001] = {'\0'};

            for (iter = 0; iter < str.size (); iter ++) {

                bufferToSend [iter] = str [iter];

            }

            zmq\_send (children [currChild].requester, bufferToSend, 1000, 0);

        }

        else if (buffer [0] == 'c') {

            char cNumber1 [11];

            int n = 0;

            while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                cNumber1 [n] = buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            long int messageSize = strtol (cNumber1, NULL, 10);

            if (messageSize == 0) {

                string word, key;

                while (

                    ((buffer [iter] >= '0') && (buffer [iter] <= '9')) ||

                    ((buffer [iter] >= 'a') && (buffer [iter] <= 'z')) ||

                    ((buffer [iter] >= 'A') && (buffer [iter] <= 'Z'))

                    ) {

                    word += buffer [iter];

                    n ++;

                    iter ++;

                }

                n = 0;

                iter ++;

                while (

                    ((buffer [iter] >= '0') && (buffer [iter] <= '9')) ||

                    ((buffer [iter] >= 'a') && (buffer [iter] <= 'z')) ||

                    ((buffer [iter] >= 'A') && (buffer [iter] <= 'Z'))

                    ) {

                    key += buffer [iter];

                    n ++;

                    iter ++;

                }

                string answer = "Ok:" + to\_string (thisNodeID);

                if (key == "n") {

                    auto res = dictionary.find (word);

                    if (res == dictionary.end ()) {

                        answer += ": \'" + word + "\' not found";

                    }

                    else {

                        answer += ": " + to\_string (res->second);

                    }

                }

                else {

                    long int keyN = strtol (key.c\_str (), NULL, 10);

                    pair <string, long int> p = {word, keyN};

                    dictionary.insert (p);

                }

                cout << answer << endl;

                continue;

            }

            char cNumber2 [11];

            while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                cNumber2 [n] = buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            long int currChild = strtol (cNumber2, NULL, 10);

            vector <long int> path;

            for (long int i = 0; i < messageSize - 1; i ++) {

                char cNumber3 [11];

                while ((buffer [iter] >= '0') && (buffer [iter] <= '9')) {

                    cNumber3 [n] = buffer [iter];

                    n ++;

                    iter ++;

                }

                n = 0;

                iter ++;

                path.push\_back (strtol (cNumber3, NULL, 10));

            }

            string text, pattern;

            while (

                ((buffer [iter] >= '0') && (buffer [iter] <= '9')) ||

                ((buffer [iter] >= 'a') && (buffer [iter] <= 'z')) ||

                ((buffer [iter] >= 'A') && (buffer [iter] <= 'Z'))

                ) {

                text += buffer [iter];

                n ++;

                iter ++;

            }

            n = 0;

            iter ++;

            while (

                ((buffer [iter] >= '0') && (buffer [iter] <= '9')) ||

                ((buffer [iter] >= 'a') && (buffer [iter] <= 'z')) ||

                ((buffer [iter] >= 'A') && (buffer [iter] <= 'Z'))

                ) {

                pattern += buffer [iter];

                n ++;

                iter ++;

            }

            string str = "c";

            str += to\_string (path.size ());

            str += ' ';

            for (int i = 0; i < path.size (); i ++) {

                str += to\_string (path [i]);

                str += ' ';

            }

            str += text;

            str += ' ';

            str += pattern;

            str += ' ';

            char bufferToSend [1001] = {'\0'};

            for (iter = 0; iter < str.size (); iter ++) {

                bufferToSend [iter] = str [iter];

            }

            zmq\_send (children [currChild].requester, bufferToSend, 1000, 0);

        }

        else {

            Sleep (1000);

        }

    }

    return 0;

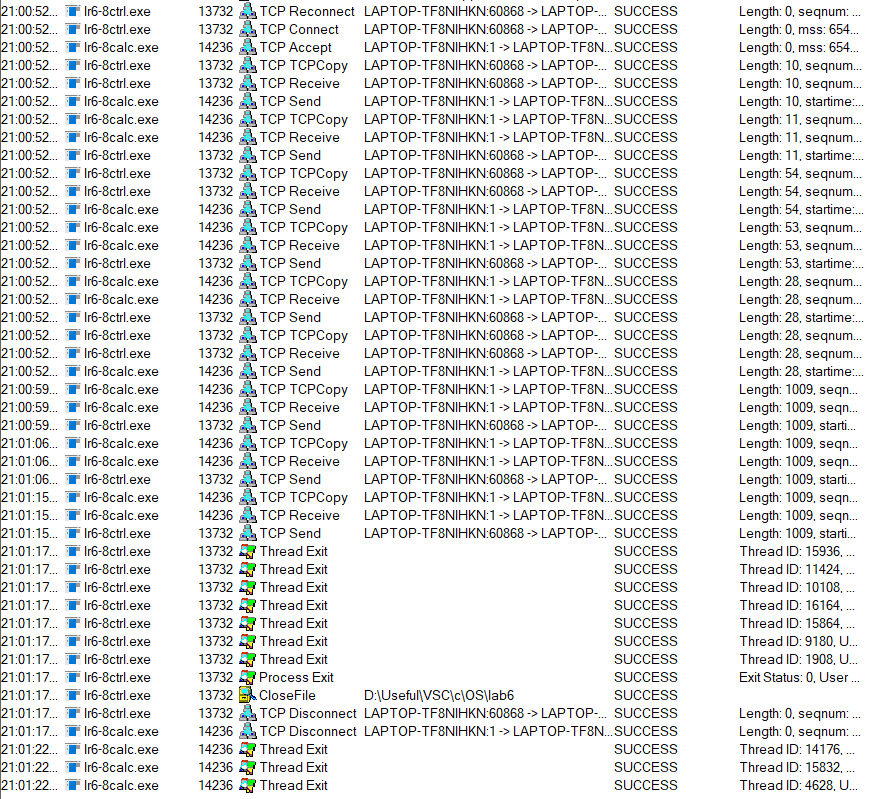
}

/\*

g++ .\lab6-calc-v3.cpp -o calc.exe -L. -llibzmq-v142-mt-gd-4\_3\_5

\*/

**ProcMon**



**Выводы**

Я изучил работу с очередями сообщений. Неожиданно, наиболее сложной частью оказалось не написание программы, а установка библиотеки ZeroMQ на компьютер, а точнее на Windows. Так как я выполнял все предыдущие лабораторные именно на этой системе, мне не хотелось переходить на Unix, который корректно работает с данной библиотекой.

**Список литературы**

1. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. —СПб.: Издательский дом «Питер», 2018. — С. 111 - 123