Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Операционные системы»

3 семестр

Задание 2

Вариант 20

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Миронов Евгений Сергеевич |
| Оценка: |  |
| Дата: | 22.11.2019 |

Москва, 2019

**Содержание**

1. **Задание**................................................................................................................2
2. **Адрес репозитория на GitHub**.........................................................................2
3. **Код программы**..................................................................................................2
4. **Результаты выполнения тестов**......................................................................9
5. **Объяснение результатов работы программы**............................................10
6. **Вывод**.................................................................................................................11

**1.Задание**

Цель работы: научиться управлять процессами и обеспечивать передачу данных между ними, посредством каналов

Задание: cоставить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 20: дочерний процесс представляет собой сервер по работе с деревом общего вида и принимает команды со стороны родительского процесса.

**2.Адрес репозитория на GitHub**

https://github.com/Dmitry4K/labOS2

**3.Код программы**

*Client:*

*client.cpp:*

#define \_CRT\_USE\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include<tchar.h>

#include<iostream>

#include"split.h"

#include<locale.h>

int StrToChar(const char\* str) {

int i = 0, res = 0;

while (str[i] != '\0') {

res \*= 10;

res += str[i] - '0';

i++;

}

return res;

}

void State(const char\* str) {

printf(">>> %s\n", str);

}

void Menu() {

printf("create [ключ] - создать дерево\n");

printf("add [ключ] [ключ] - добавить узел\n");

printf("del [ключ] - удалить узел\n");

printf("clear - очистить дерево\n");

printf("print - распечатать дерево\n");

printf("exit - выйти\n");

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

SECURITY\_ATTRIBUTES sa;

sa.nLength = sizeof(SECURITY\_ATTRIBUTES);

sa.lpSecurityDescriptor = NULL;

sa.bInheritHandle = TRUE;

PROCESS\_INFORMATION ProcessInfo; //This is what we get as an [out] parameter

ZeroMemory(&ProcessInfo, sizeof(PROCESS\_INFORMATION));//обнулить ProcessInfo

STARTUPINFO StartupInfo; //This is an [in] parameter

TCHAR lpszClientPath[] = "server";//название процесса

ZeroMemory(&StartupInfo, sizeof(StartupInfo));

StartupInfo.cb = sizeof(STARTUPINFO); //Only compulsory field

HANDLE pipe1Read, pipe1Write, pipe2Read, pipe2Write;//идентификатор объекта

CreatePipe(&pipe1Read, &pipe1Write, &sa, 0);

CreatePipe(&pipe2Read, &pipe2Write, &sa, 0);

StartupInfo.dwFlags = STARTF\_USESTDHANDLES;

StartupInfo.hStdInput = pipe1Read;

StartupInfo.hStdOutput = pipe2Write;

bool process = CreateProcess(NULL,

lpszClientPath,//процесс

NULL, NULL, true,

CREATE\_NO\_WINDOW,// CREATE\_NEW\_CONSOLE|CREATE\_SUSPENDED

NULL, NULL,

&StartupInfo,

&ProcessInfo);

process ? State("процесс создан") : State("ошибка: процесс не создан");

CloseHandle(pipe1Read);

CloseHandle(pipe2Write);

DWORD writeBytes, readBytes;

char masstr[256];

char \*str;

printf("\n");

Menu();

printf("\n");

while (1) {

str = gets\_s(masstr, 256);

char \*\*commands = split(str, ' ');

bool isSuccess;

int i = 0, num;

while (commands[i][0] != '\0') {

if (!strcmp(commands[i], "create")) {

num = 0;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено create") : State("Команда не отправлена");

i++;

num = StrToChar(commands[i]);

WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

}

else if (!strcmp(commands[i], "add")) {

num = 1;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено add") : State("Команда не отправлена");

i++;

num = StrToChar(commands[i]);

WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

i++;

num = StrToChar(commands[i]);

WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

}

else if (!strcmp(commands[i], "del")) {

num = 2;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено del") : State("Команда не отправлена");

i++;

num = StrToChar(commands[i]);

WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

}

else if (!strcmp(commands[i], "clear")) {

num = 3;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено clear") : State("Команда не отправлена");

}

else if (!strcmp(commands[i], "print")) {

num = 4;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено print") : State("Команда не отправлена");

char c = '0';

while (c != '\0') {

ReadFile(pipe2Read, &c, sizeof(char), &readBytes, NULL);

printf("%c", c);

}

printf("\b");

}

else if (!strcmp(commands[i], "exit")) {

num = 5;

isSuccess = WriteFile(pipe1Write, &num, sizeof(int), &writeBytes, NULL);

isSuccess ? State("отправлено exit") : State("Команда не отправлена");

CloseHandle(pipe2Read);

CloseHandle(pipe1Write);

CloseHandle(ProcessInfo.hThread);

CloseHandle(ProcessInfo.hProcess);

system("pause");

return 0;

}

i++;

}

}

CloseHandle(pipe2Read);

CloseHandle(pipe1Write);

CloseHandle(ProcessInfo.hThread);

CloseHandle(ProcessInfo.hProcess);

system("pause");

return 0;

}

*split.h:*

#pragma once

char\*\* split(char \* str, char sep);

*split.cpp:*

#pragma once

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include"split.h"

char\*\* split(char\* str, char sep) {

char\*\* res = nullptr;

if (res == nullptr) {

if (str == nullptr)

return res;

int str\_size = -1, substring\_count = 0, i = 0, j = 0, was\_sep = 1, substring\_length = 0, start\_index = 0;

do {

str\_size++;

if ((str[str\_size] == sep || str[str\_size] == '\0') && was\_sep == 0) {

was\_sep = 1;

substring\_count++;

}

else if (str[str\_size] != sep && str[str\_size] != '\0')

was\_sep = 0;

} while (str[str\_size] != '\0');

str\_size++;

if (substring\_count == 0) return res;

substring\_count++;

//printf("count:%d size:%d len:%d\n", substring\_count, str\_size, strlen(str));

res = (char\*\*)malloc(substring\_count \* sizeof(char\*));

//res = new char\*[substring\_count];

res[substring\_count - 1] = (char\*)malloc(sizeof(char));

//res[substring\_count - 1] = new char[1];

res[substring\_count - 1][0] = '\0';

for (int i = 0; i < str\_size; i++) {

// printf("i:%d\n", i);

substring\_length++;

if ((str[i] == sep || str[i] == '\0') && substring\_length > 1) {

// printf("sub\_length:%d j:%d ", substring\_length, j);

res[j] = (char\*)malloc(substring\_length \* sizeof(char));

// res[j] = new char[substring\_length];

for (int k = i - substring\_length + 1, l = 0; k < i; ++k, ++l) {

res[j][l] = str[k];

}

res[j][substring\_length - 1] = '\0';

substring\_length = 0;

j++;

}

else if (str[i] == sep || str[i] == '\0') substring\_length = 0;

}

}

return res;

}

*Server:*

*server.cpp:*

#include<Windows.h>

#include<tchar.h>

#include"CTree.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

HANDLE inH = GetStdHandle(STD\_INPUT\_HANDLE);

HANDLE outH = GetStdHandle(STD\_OUTPUT\_HANDLE);

DWORD readBytes;

cTree\* ctree = nullptr;

int a,b,c;

while (1) {

ReadFile(inH, &a, sizeof(int), &readBytes, 0);

switch (a) {

case 0:

ReadFile(inH, &b, sizeof(int), &readBytes, 0);

ctree = cTreeCreate(b);

break;

case 1:

ReadFile(inH, &b, sizeof(int), &readBytes, 0);

ReadFile(inH, &c, sizeof(int), &readBytes, 0);

cTreeAddNode(ctree, b, c);

break;

case 2:

ReadFile(inH, &b, sizeof(int), &readBytes, 0);

cTreeDeleteNode(ctree, b);

break;

case 3:

cTreeDestroy(ctree);

break;

case 4:

cTreePrintTo(ctree, outH);

break;

case 5:

return 0;

}

}

return 0;

}

*CTree.h:*

#pragma once

#include<Windows.h>

struct cNode {

int key;

cNode\* parent;

cNode\* son;

cNode\* brother;

};

struct cTree {

cNode\* root;

};

cTree\* cTreeCreate(int key);//проверено

cNode\* cTreeFindNodeByKey(cNode\* tree, int key);//проверено

void cTreeAddNode(cTree\* tree, int to, int key);//проверено

void cTreeDeleteNode(cTree\* tree, int key);

void cTreeClear(cNode\* node);//проверено

void cTreeDestroy(cTree\* tree);//проверено

void cTreePrint(cTree\* tree);

void cTreePrint\_(cNode\* node, int count);

void cTreePrintTo(cTree\* tree, HANDLE outH);

void cTreePrintTo\_(cNode\* node, int count, HANDLE outH, DWORD\* writebytes);

*CTree.cpp:*

#include<malloc.h>

#include<iostream>

#include"CTree.h"

#include<Windows.h>

cTree\* cTreeCreate(int key) {

cTree\* res = (cTree\*)malloc(sizeof(cTree));

res->root = (cNode\*)malloc(sizeof(cNode));

res->root->key = key;

res->root->parent = nullptr;

res->root->brother = nullptr;

res->root->son = nullptr;

return res;

}

cNode\* cTreeFindNodeByKey(cNode\* node, int key) {

cNode\* res = nullptr;

if (node->key == key)

return node;

if (node->son)

res = cTreeFindNodeByKey(node->son, key);

if (node->brother && !res)

res = cTreeFindNodeByKey(node->brother, key);

return res;

}

void cTreeAddNode(cTree\* tree, int to, int key) {

cNode\* fnode = nullptr;

fnode = cTreeFindNodeByKey(tree->root, to);

if (!fnode) {

return;

}

if (!fnode->son) {

fnode->son = (cNode\*)malloc(sizeof(cNode));

fnode->son->key = key;

fnode->son->parent = fnode;

fnode->son->brother = nullptr;

fnode->son->son = nullptr;

return;

}

else {

cNode\* bnode = nullptr;

bnode = fnode->son;

while (bnode->brother)

bnode = bnode->brother;

bnode->brother = (cNode\*)malloc(sizeof(cNode));

bnode->brother->key = key;

bnode->brother->parent = fnode;

bnode->brother->brother = nullptr;

bnode->brother->son = nullptr;

return;

}

return;

}

void cTreeDeleteNode(cTree\* tree, int key) {

cNode\* fnode = nullptr;

fnode = cTreeFindNodeByKey(tree->root, key);

if (!fnode)

return;

if (fnode->son)

cTreeClear(fnode->son);

cNode\* inode = fnode->parent->son;

if (inode == fnode) {

if (fnode->brother)

fnode->parent->son = fnode->brother;

else

fnode->parent->son = nullptr;

free(fnode);

}

else {

while (inode->brother != fnode) {

inode = inode->brother;

}

if (fnode->brother) {

inode->brother = fnode->brother;

free(fnode);

}

else {

inode->brother = nullptr;

free(fnode);

}

}

}

void cTreeClear(cNode\* node) {

if (node->son)

cTreeClear(node->son);

if (node->brother)

cTreeClear(node->brother);

node->brother = nullptr;

node->son = nullptr;

if (node->parent)

if (node->parent->son == node)

node->parent->son = nullptr;

free(node);

}

void cTreeDestroy(cTree\* tree) {

cTreeClear(tree->root);

tree->root = nullptr;

}

void cTreePrint(cTree\* tree) {

if (tree->root)

cTreePrint\_(tree->root, 0);

}

void cTreePrint\_(cNode\* node, int count) {

for (int i = 0; i < count; i++)

printf("\t");

printf("%d\n", node->key);

if (node->son)

cTreePrint\_(node->son, count + 1);

if (node->brother)

cTreePrint\_(node->brother, count);

}

void cTreePrintTo(cTree\* tree, HANDLE outH) {

DWORD writebytes = 0;

char c = '\0';

if (tree->root)

cTreePrintTo\_(tree->root, 0, outH, &writebytes);

WriteFile(outH, &c, sizeof(char), &writebytes, 0);

}

void cTreePrintTo\_(cNode\* node, int count, HANDLE outH, DWORD\* writebytes) {

char c = '\t';

for (int i = 0; i < count; i++)

WriteFile(outH, &c, sizeof(char), writebytes, 0);

c = node->key + '0';

WriteFile(outH, &c, sizeof(char), writebytes, 0);

c = '\n';

WriteFile(outH, &c, sizeof(char), writebytes, 0);

if (node->son)

cTreePrintTo\_(node->son, count + 1, outH, writebytes);

if (node->brother)

cTreePrintTo\_(node->brother, count, outH, writebytes);

}

**4.Результаты выполнения тестов**

>>> процесс создан

create [ключ] - создать дерево

add [ключ] [ключ] - добавить узел

del [ключ] - удалить узел

clear - очистить дерево

print - распечатать дерево

exit - выйти

create 0 add 0 1 add 0 2 add 1 3

>>> отправлено create

>>> отправлено add

>>> отправлено add

>>> отправлено add

print

>>> отправлено print

0

1

3

2

del 1

>>> отправлено del

print

>>> отправлено print

0

2

clear

>>> отправлено clear

print

>>> отправлено print

exit

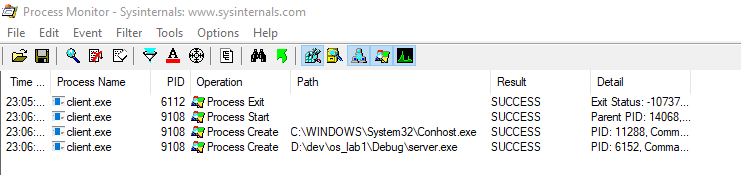
>>> отправлено exit

Для продолжения нажмите любую клавишу . . .

**5.Объяснение результатов работы программы**

В своей работе программа используется два процесса. 1ый отвечает за чтение команд и их отправку на сервер, который является вторым процессом, на самом сервере, исходя из полученных команд происходят соответствующие манипуляции над деревом общего вида, в случае если требуется вывести дерево, оно посимвольно записывает в канал, с которого также посимвольно считывается дерево представленное в определенном формате. Для организации данного клиент-серверного приложения, требовалось создать процесс и организовать два канала.

Убедиться в том, что программа работает корректно можно используя программу Process Monitor.



**6.Вывод**

По итогу выполнения данной лабораторной работы узнал, как работают многопроцессорные приложения, а также как происходит общения между процессами в таких приложениях. В будущем данные знания будут крайне полезны, так любое приложение, имеющее интерфейс или огромный ряд функций и возможностей, использует в себе сразу несколько процессов.