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 4

Вариант 20

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Миронов Евгений Сергеевич |
| Оценка: |  |
| Дата: | 17.12.2019 |

Москва, 2019

**Содержание**

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

**1.Задание**

Составить 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 через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант: 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/labOS4

**3.Код программы**

*Clinet.cpp*

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

#define EVENT\_NAME\_FIRST L"Local\\FirstEvent"

#define EVENT\_NAME\_SECOND L"Local\\SecondEvent"

#define MAP\_LOCATION L"Local\\FileMapObject"

#define MUTEX\_NAME L"Local\\newmutex"

#include <Windows.h>

#include<tchar.h>

#include<iostream>

#include"split.h"

#include<locale.h>

#include<string>

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[] = L"server";//название процесса

ZeroMemory(&StartupInfo, sizeof(StartupInfo));

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

HANDLE hFileMap = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_EXECUTE\_READWRITE, 0, 256, MAP\_LOCATION);

if (hFileMap == INVALID\_HANDLE\_VALUE) {

State("Error: file not mapped\n");

system("pause");

return 0;

}

PCHAR lbBuffer = (PCHAR)MapViewOfFile(hFileMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, 256);

if (lbBuffer == nullptr) {

State("Error reading map");

system("pause");

return 0;

}

HANDLE hEvent1 = CreateEvent(NULL, FALSE, FALSE, EVENT\_NAME\_FIRST);

HANDLE hEvent2 = CreateEvent(NULL, FALSE, FALSE, EVENT\_NAME\_SECOND);

/\*

HANDLE hMutex = CreateMutex(NULL, TRUE, MUTEX\_NAME);//залоченый мютекс

if (hMutex == 0) {

std::cout << "Creating Mutex Error : " << GetLastError()<<std::endl;

}

\*/

bool process = CreateProcess(NULL,lpszClientPath,NULL, NULL, TRUE,CREATE\_NEW\_CONSOLE,NULL, NULL,&StartupInfo,&ProcessInfo);

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

DWORD writeBytes, readBytes;

//ar masstr[256];

//char \*str = nullptr;

printf("\n");

Menu();

printf("\n");

std::string line;

bool isExit = false;

int i = 0;

while (true){

getline(std::cin, line);

char\*\* comands = split(const\_cast<char\*>(line.data()), ' ');

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

if (!strcmp(comands[i++], "exit"))

isExit = true;

}

i = 0;

CopyMemory(lbBuffer, line.data(), 256);

//ReleaseMutex(hMutex);

ResetEvent(hEvent1);

SetEvent(hEvent2);

WaitForSingleObject(hEvent1,INFINITE);

std::cout << lbBuffer;

if (isExit)

break;

}

//UnmapViewOfFile(MAP\_LOCATION);

CloseHandle(ProcessInfo.hThread);

CloseHandle(ProcessInfo.hProcess);

CloseHandle(hEvent1);

CloseHandle(hEvent2);

return 0;

}

*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;

}

*Split.h*

#pragma once

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

*Server.cpp*

#define MAP\_LOCATION L"Local\\FileMapObject"

#define EVENT\_NAME\_FIRST L"Local\\FirstEvent"

#define EVENT\_NAME\_SECOND L"Local\\SecondEvent"

#define MUTEX\_NAME L"Local\\newmutex"

#include<Windows.h>

#include<tchar.h>

#include"CTree.h"

#include<iostream>

#include"split.h"

int StrToChar(const char\* str) {

int i = 0, res = 0;

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

res \*= 10;

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

i++;

}

return res;

}

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

{

/\*

HANDLE hMutex = OpenMutex(

MUTEX\_ALL\_ACCESS, // default security descriptor

FALSE, // mutex not owned

MUTEX\_NAME);

if (hMutex == 0) {

std::cout << "Mutex Error\n"<<GetLastError();

} \*/

HANDLE hEvent1 = OpenEvent(EVENT\_ALL\_ACCESS,FALSE, EVENT\_NAME\_FIRST);

HANDLE hEvent2 = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, EVENT\_NAME\_SECOND);

if (hEvent1 == 0) {

std::cout << "Event error: " << GetLastError() << std::endl;

system("pause");

return 1;

}

if (hEvent2 == 0) {

std::cout << "Event error: " << GetLastError() << std::endl;

system("pause");

return 1;

}

cTree\* ctree = nullptr;

HANDLE hFileMap = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, TRUE, MAP\_LOCATION);

if (hFileMap == INVALID\_HANDLE\_VALUE) {

std::cout << "Error filemapping\n";

}

while (1) {

//WaitForSingleObject(hMutex, INFINITE);

WaitForSingleObject(hEvent2, INFINITE);

//Sleep(2000);

PCHAR lbBuffer = (PCHAR)MapViewOfFile(hFileMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, 256);

if (lbBuffer == nullptr) {

std::cout << "Error reading map\n";

}

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

bool NeedClear = true;

int i = 0, a,b,c;

std::string buffer;

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

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

std::cout << "create\n";

i++;

a = StrToChar(commands[i]);

ctree = cTreeCreate(a);

}

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

std::cout << "add\n";

i++;

b = StrToChar(commands[i]);

i++;

c = StrToChar(commands[i]);

cTreeAddNode(ctree, b, c);

}

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

std::cout << "del\n";

i++;

b = StrToChar(commands[i]);

cTreeDeleteNode(ctree, b);

}

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

std::cout << "clear\n";

cTreeDestroy(ctree);

}

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

ZeroMemory(lbBuffer, sizeof(PCHAR));

NeedClear = false;

std::cout << "print\n";

buffer = cTreePrintToPtr(ctree);

std::cout << buffer;

CopyMemory(lbBuffer, buffer.data(), 256);

//printf("\b");

}

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

ZeroMemory(lbBuffer, sizeof(PCHAR));

//(hMutex);

SetEvent(hEvent1);

return 0;

}

i++;

}

if(NeedClear)

ZeroMemory(lbBuffer, sizeof(PCHAR));

ResetEvent(hEvent2);

SetEvent(hEvent1);

//ReleaseMutex(hMutex);

}

system("pause");

return 0;

}

*CTree.cpp*

#include<malloc.h>

#include<iostream>

#include"CTree.h"

#include<Windows.h>

#include<string>

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);

}

void cTreePrintToPtr\_(cNode\* node, int count, std::string\* res) {

char c = '\t';

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

(\*res) += c;

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

c = node->key + '0';

(\*res) += c;

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

c = '\n';

(\*res) += c;

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

if (node->son)

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

if (node->brother)

cTreePrintToPtr\_(node->brother, count, res);

}

std::string cTreePrintToPtr(cTree\* tree) {

//char c = '\0';

std::string res;

if (tree->root)

cTreePrintToPtr\_(tree->root, 0, &res);

res += '\0';

return res;

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

}

*CTree.h*

#pragma once

#include<Windows.h>

#include<string>

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);

std::string cTreePrintToPtr(cTree\* tree);

void cTreePrintToPtr\_(cNode\* node,int count,std::string\* res);

**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 print

0

1

add 1 2 add 0 3

print

0

1

2

3

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, всё общение между процессорами происходит за счет file mapping.

Для синхронизации работы двух процессоров используются WinApi события, в моей программе их два, и если рассматривать их с точки зрения обычного человека, то события- это своего рода рычаги, которые либо открывают дверь либо – блокируют.

**6.Вывод**

Технология file mapping исходя из результатов лабораторной работы оказалась менее удобная в использовании чем каналы, так как требуется постоянно следить за содержимым файла, следить за тем, как записывать в файл новые данные (следить за положением указателя в файле). В тоже время работа с событиями оказалась простой и интуитивно понятной, намного удобнее и приятнее пользоваться событиями и напрямую «говорить» программе, когда она должна выполняться, по моему мнению – события один из лучших примитивов синхронизации.