Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №6

на тему

СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ (WINDOWS). ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕ СРЕДСТВ СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ.

Выполнил студент гр.153502 Леоненко А.О.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Теоретические сведения 3](#_Toc151124244)

[2 Формулировка задачи 4](#_Toc151124245)

[3 Описание функций программы 5](#_Toc151124246)

[3.1 Результат работы программы 5](#_Toc151124247)

[Список использованных источников 6](#_Toc151124248)

[Приложение А 7](#_Toc151124249)

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Объект синхронизации – это объект, дескриптор которого можно указать в одной из функций ожидания для координации выполнения нескольких потоков. Несколько процессов могут иметь дескриптор одного и того же объекта синхронизации, что делает возможной синхронизацию между процессами.

Объект события полезен при отправке в поток сигнала, указывающего на то, что произошло определенное событие. Например, в перекрывающихся входных и выходных данных система присваивает указанному объекту события сигнальное состояние после завершения перекрывающейся операции. Один поток может указать различные объекты событий в нескольких одновременных перекрывающихся операциях, а затем использовать одну из функций ожидания с несколькими объектами, чтобы дождаться получения сигнала о состоянии любого из объектов событий.

Поток использует функцию *CreateEvent* или *CreateEventEx* для создания объекта события. Создаваемый поток определяет начальное состояние объекта и указывает, является ли он объектом события сброса вручную или автоматическим сбросом. Создающий поток также может указать имя объекта события. Потоки в других процессах могут открывать дескриптор для существующего объекта события, указывая его имя в вызове функции *OpenEvent.*

Объект мьютекса – это объект синхронизации, состояние которого устанавливается в значение *Signaled*, если он не принадлежит ни одному потоку, и без знака, когда он принадлежит. Только один поток за раз может владеть объектом мьютекса, имя которого происходит из-за того, что он полезен для координации взаимоисключающего доступа к общему ресурсу. Например, чтобы предотвратить запись двух потоков в общую память одновременно, каждый поток ожидает владения объектом мьютекса перед выполнением кода, который обращается к памяти. После записи в общую память поток освобождает объект мьютекса.

Поток использует функцию *CreateMutex* или *CreateMutexEx* для создания объекта мьютекса. Создающий поток может запросить немедленное владение объектом мьютекса, а также указать имя для объекта мьютекса. Он также может создать неименованный мьютекс. [1]

2 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение и использование средств синхронизации и взаимного исключения.

В качестве задачи необходимо создать многозадачное приложение, использующее средства синхронизации (мьютексы или семафоры) для координирования работы потоков.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

1 мониторинг изменений реестра;

2 запуск мониторинга нескольких реестров;

3 использование *mutex* для синхронизации потоков.

3.1 Результат работы программы

На рисунке 3.1 изображен результат работы программы. При запуске в консоль выводится информация о текущих значениях выбранных реестров. При изменении в реестре, в консоль выводится информация об этом изменении. При изменении в любом реестре выводится общее количество изменений.

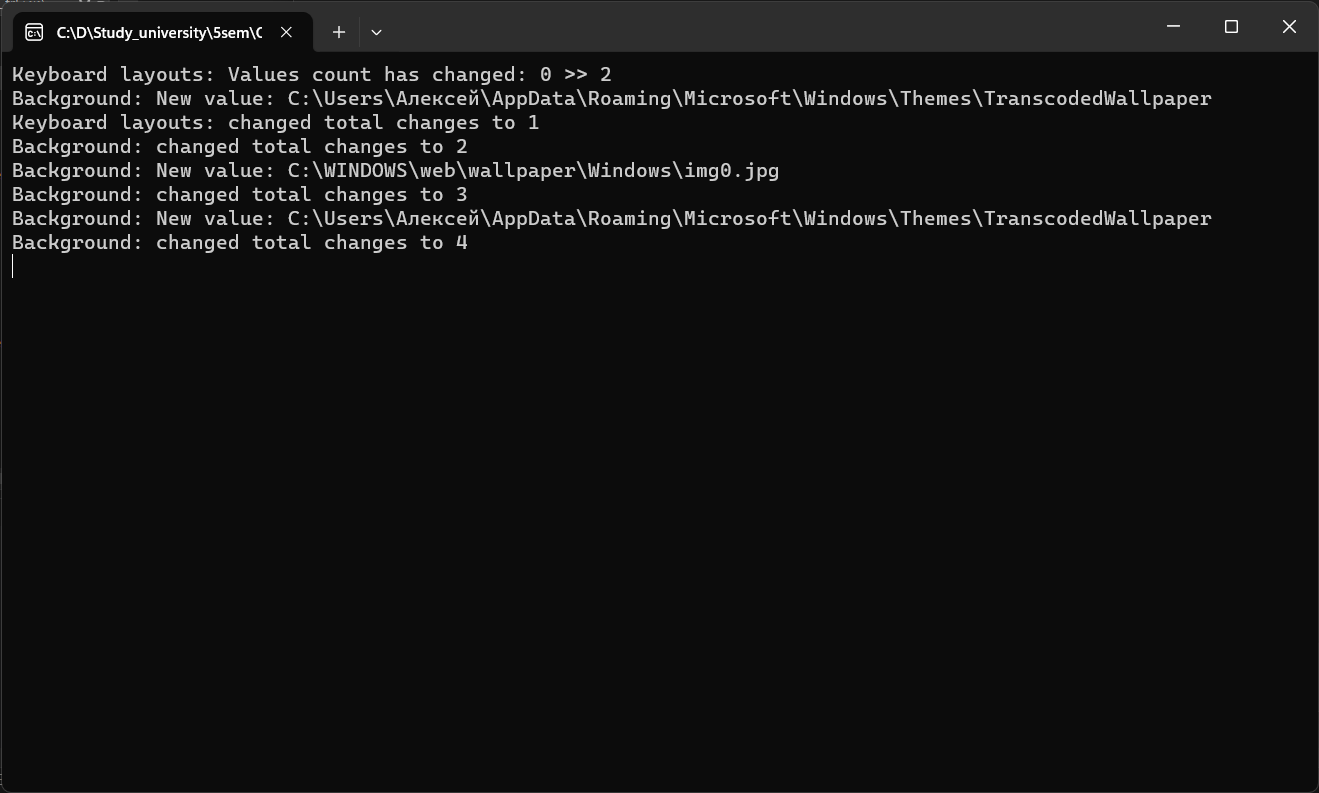


Рисунок 3.1 – Результат работы программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Объекты синхронизации – Win32 apps | Microsoft Learn [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects

ПРИЛОЖЕНИЕ А

Листинг кода

Файл main.cpp

#include <windows.h>

#include <iostream>

#include <locale>

HANDLE mutex;

HANDLE outputMutex;

int totalChanges = 0;

struct Args

{

HKEY hKey;

LPCWSTR subKey;

LPCWSTR valueName;

LPCWSTR taskName;

};

void ChangeTotalChanges(LPCWSTR taskName) {

DWORD waitResult;

waitResult = WaitForSingleObject(

mutex, // handle to mutex

INFINITE); // no time-out interval

switch (waitResult)

{

case WAIT\_OBJECT\_0:

\_\_try {

totalChanges++;

WaitForSingleObject(

outputMutex, // handle to mutex

INFINITE); // no time-out interval

std::wcout << taskName << L": changed total changes to " << totalChanges << std::endl;

ReleaseMutex(outputMutex);

}

\_\_finally {

ReleaseMutex(mutex);

}

break;

// The thread got ownership of an abandoned mutex

case WAIT\_ABANDONED:

return;

}

}

void MonitorRegistryChanges(LPVOID lpParam) {

Args\* args = static\_cast<Args\*>(lpParam);

HKEY hKey = args->hKey;

LPCWSTR subKey = args->subKey;

LPCWSTR valueName = args->valueName;

LPCWSTR taskName = args->taskName;

HANDLE hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

if (hEvent == NULL) {

std::cerr << "Failed to create event. Error code: " << GetLastError() << std::endl;

return;

}

RegNotifyChangeKeyValue(hKey, TRUE, REG\_NOTIFY\_CHANGE\_LAST\_SET, hEvent, TRUE);

DWORD lastValueSize = 0;

WCHAR lastValue[MAX\_PATH] = L"";

DWORD lastValueCount = 0;

while (true) {

DWORD waitResult = WaitForSingleObject(hEvent, INFINITE);

if (waitResult == WAIT\_OBJECT\_0) {

DWORD dataSize;

WCHAR data[MAX\_PATH];

if (wcscmp(valueName, L"") == 0)

{

LONG result = RegOpenKeyEx(HKEY\_CURRENT\_USER, subKey, 0, KEY\_READ, &hKey);

if (result == ERROR\_SUCCESS) {

DWORD currentValueCount;

if (RegQueryInfoKey(hKey, nullptr, nullptr, nullptr, nullptr, nullptr, nullptr, &currentValueCount, nullptr, nullptr, nullptr, nullptr) == ERROR\_SUCCESS) {

if (currentValueCount != lastValueCount) {

WaitForSingleObject(

outputMutex, // handle to mutex

INFINITE); // no time-out interval

std::wcout << taskName << L": Values count has changed: " << lastValueCount << " >> " << currentValueCount << std::endl;

ReleaseMutex(outputMutex);

lastValueCount = currentValueCount;

ChangeTotalChanges(taskName);

}

}

}

RegCloseKey(hKey);

}

else if (RegGetValue(hKey, subKey, valueName, RRF\_RT\_REG\_SZ, NULL, data, &dataSize) == ERROR\_SUCCESS) {

if (wcscmp(data, lastValue) != 0) {

WaitForSingleObject(

outputMutex, // handle to mutex

INFINITE); // no time-out interval

std::wcout << taskName << L": New value: " << data << std::endl;

ReleaseMutex(outputMutex);

wcscpy\_s(lastValue, data);

ChangeTotalChanges(taskName);

}

}

RegNotifyChangeKeyValue(hKey, TRUE, REG\_NOTIFY\_CHANGE\_LAST\_SET, hEvent, TRUE);

}

else if (waitResult == WAIT\_FAILED) {

std::cerr << "WaitForSingleObject failed. Error code: " << GetLastError() << std::endl;

break;

}

}

CloseHandle(hEvent);

}

int main() {

std::setlocale(LC\_ALL, "Russian");

mutex = CreateMutex(

NULL, // default security attributes

FALSE, // initially not owned

NULL); // unnamed mutex

outputMutex = CreateMutex(

NULL, // default security attributes

FALSE, // initially not owned

NULL); // unnamed mutex

Args\* args1 = new Args;

Args\* args2 = new Args;

args1->hKey = HKEY\_CURRENT\_USER;

args1->subKey = L"Control Panel\\Desktop";

args1->valueName = L"Wallpaper";

args1->taskName = L"Background";

args2->hKey = HKEY\_CURRENT\_USER;

args2->subKey = L"Keyboard Layout\\Preload";

args2->valueName = L"";

args2->taskName = L"Keyboard layouts";

HANDLE thread1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)MonitorRegistryChanges, (LPVOID)args1, 0, NULL);

HANDLE thread2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)MonitorRegistryChanges, (LPVOID)args2, 0, NULL);

WaitForSingleObject(thread1, INFINITE);

WaitForSingleObject(thread2, INFINITE);

return 0;

}