Пример с мьютексами

```
#include <iostream>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

HANDLE hMutex;

void ThreadPlus(char expression[50][10])
{
    WaitForSingleObject(hMutex, INFINITE);
    for (int i = 0; expression[i][0] != '\0'; i++){
        if (!strcmp(expression[i], "+") || !strcmp(expression[i], "-")){
    std::cout << "\nНашли + или - ---> expression[i] = " << expression[i] <<
"\n";
            int tempNumber;
            if (!strcmp(expression[i], "+")){
            tempNumber = atoi(expression[i - 1]) + atoi(expression[i + 1]);
            }
            else if (!strcmp(expression[i], "-")){
            tempNumber = atoi(expression[i - 1]) - atoi(expression[i + 1]);
            }
            //копируем результат вычисления в ячейку i - 1
            _itoa_s(tempNumber, expression[i - 1], 10);
            //смешение всех знаков или чисел на два знака влево
            int j;
            for (j = i; expression[j + 2][0] != '\0'; j++){
                strcpy_s(expression[j], expression[j + 2]);
            }
            strcpy_s(expression[j], "");
            strcpy_s(expression[j + 1], "");
            std::cout << "\nВывод-промежуточно\n";
            for (int i = 0; expression[i][0] != '\0'; i++){
                std::cout << expression[i];
            }
            //откатываемся назад для проверки
            i--;
        }

    }
    ReleaseMutex(hMutex);
}
void ThreadMultiply(char expression[50][10])
{
    WaitForSingleObject(hMutex, INFINITE);
    for (int i = 0; expression[i][0] != '\0'; i++){
        if (!strcmp(expression[i], "*") || !strcmp(expression[i], "/")){
    std::cout << "\nНашли * или / ---> expression[i] = " << expression[i] <<
"\n";
            int tempNumber;
            if (!strcmp(expression[i], "*")){
            tempNumber = atoi(expression[i - 1]) * atoi(expression[i + 1]);
            }
            else if (!strcmp(expression[i], "/")){
            tempNumber = atoi(expression[i - 1]) / atoi(expression[i + 1]);
            }

            _itoa_s(tempNumber, expression[i - 1], 10);
            //смешение всех знаков или чисел на два знака влево
            int j;
            for (j = i; expression[j + 2][0] != '\0'; j++){
```

```
                strcpy_s(expression[j], expression[j + 2]);
            }
            strcpy_s(expression[j], "");
            strcpy_s(expression[j + 1], "");
            std::cout << "\nВывод-промежуточно\n";
            for (int i = 0; expression[i][0] != '\0'; i++){
                std::cout << expression[i];
            }
            //откатываемся назад для проверки
            i--;
        }
        Sleep(100);     //Показать, что МЬЮТЕКС РАБОТАЕТ
    }
    ReleaseMutex(hMutex);
}

int main(void)
{
#pragma warning(disable : 4996)
    setlocale(LC_ALL, "Russian");
    system("color f0");
    //тестовый массив
char expression[50][10] = { "100", "+", "22", "+", "9", "+", "10", "*", "3",
"*", "2", "+", "3", "-", "10", "/", "5" };
    std::cout << "Initinal expression:\n\n";
    for (int i = 0; expression[i][0] != '\0'; i++){
        std::cout << expression[i];
    }
    std::cout << "\n";
    HANDLE hThrPlus;
    HANDLE hThrMultiply;
    const int COUNT_THREAD = 2;
    HANDLE arrayHandleThread[COUNT_THREAD];
    unsigned long uThrPlusID;
    unsigned long uThrMultiplyID;
    hMutex = CreateMutex(NULL, FALSE, NULL);
    hThrMultiply = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)ThreadMultiply, expression, 0, &uThrMultiplyID);
    if (hThrMultiply != NULL){
        std::cout << "\n1-ый поток создался\n";
hThrPlus = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadPlus,
expression, 0, &uThrPlusID);
        if (hThrPlus){
            std::cout << "\n2-ой поток создался\n";
        }
        else{
            std::cout << "\n2-ой поток не удалось создать\n";
        }
    }
    else{
    std::cout << "\n1-ый поток не удалось создать! Поэтому не будем создавать
2-ой!";
    }
//заносим в массив дескрипторов, чтобы отследить, когда потоки завершат свою
работу
    arrayHandleThread[0] = hThrMultiply;
    arrayHandleThread[1] = hThrPlus;
    //ждем когда потоки завершат свою работу
    WaitForMultipleObjects(COUNT_THREAD, arrayHandleThread, TRUE, INFINITE);
    std::cout << "\n_____________Вывод_____________\n";
    for (int i = 0; expression[i][0] != '\0'; i++){

        std::cout << expression[i];
    }
```

```
    //закрываем потоки
    CloseHandle(hThrMultiply);
    CloseHandle(hThrPlus);
    CloseHandle(hMutex);
    std::cout << "\n";
    system("pause");
    return 0;
}
```

Пример с событиями

```
#include <iostream>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

HANDLE hEvent;

void ThreadPlus(char expression[50][10])
{
    WaitForSingleObject(hEvent, INFINITE);
    //устанавливаем в значение занято
    ResetEvent(hEvent);
    for (int i = 0; expression[i][0] != '\0'; i++){
        if (!strcmp(expression[i], "+") || !strcmp(expression[i], "-")){
    std::cout << "\nНашли + или - ---> expression[i] = " << expression[i] <<
"\n";
            int tempNumber;
            if (!strcmp(expression[i], "+")){
            tempNumber = atoi(expression[i - 1]) + atoi(expression[i + 1]);
            }
            else if (!strcmp(expression[i], "-")){
            tempNumber = atoi(expression[i - 1]) - atoi(expression[i + 1]);
            }
            //копируем результат вычисления в ячейку i - 1
            _itoa_s(tempNumber, expression[i - 1], 10);
            //смешение всех знаков или чисел на два знака влево
            int j;
            for (j = i; expression[j + 2][0] != '\0'; j++){
                strcpy_s(expression[j], expression[j + 2]);
            }
            strcpy_s(expression[j], "");
            strcpy_s(expression[j + 1], "");
            std::cout << "\nВывод-промежуточно\n";
            for (int i = 0; expression[i][0] != '\0'; i++){
                std::cout << expression[i];
            }
            //откатываемся назад для проверки
            i--;
        }

    }
    SetEvent(hEvent);   //переводим в положение свободно
}

void ThreadMultiply(char expression[50][10])
{
    WaitForSingleObject(hEvent, INFINITE);
    //устанавливаем в значение занято
    ResetEvent(hEvent);
    for (int i = 0; expression[i][0] != '\0'; i++){
```

```
        if (!strcmp(expression[i], "*") || !strcmp(expression[i], "/")){
            std::cout << "\nНашли * или / ---> expression[i] = " <<
expression[i] << "\n";
            int tempNumber;
            if (!strcmp(expression[i], "*")){
            tempNumber = atoi(expression[i - 1]) * atoi(expression[i + 1]);
            }
            else if (!strcmp(expression[i], "/")){
            tempNumber = atoi(expression[i - 1]) / atoi(expression[i + 1]);
            }
            _itoa_s(tempNumber, expression[i - 1], 10);
            //смешение всех знаков или чисел на два знака влево
            int j;
            for (j = i; expression[j + 2][0] != '\0'; j++){
                strcpy_s(expression[j], expression[j + 2]);
            }
            strcpy_s(expression[j], "");
            strcpy_s(expression[j + 1], "");
            std::cout << "\nВывод-промежуточно\n";
            for (int i = 0; expression[i][0] != '\0'; i++){
                std::cout << expression[i];
            }
            //откатываемся назад для проверки
            i--;
        }
        Sleep(100);     //Показать, что МЬЮТЕКС РАБОТАЕТ (поток спит, но не
передает управлнеие другому)
    }
    SetEvent(hEvent);   //переводим в положение свободно
}

int main(void)
{
#pragma warning(disable : 4996)
    setlocale(LC_ALL, "Russian");
    system("color f0");
    //тестовый массив
    char expression[50][10] = { "100", "+", "22", "+", "9", "+", "10", "*",
"3", "*", "2", "+", "3", "-", "10", "/", "5" };
    std::cout << "Initinal expression:\n\n";
    for (int i = 0; expression[i][0] != '\0'; i++){
        std::cout << expression[i];
    }
    std::cout << "\n";
    HANDLE hThrPlus;
    HANDLE hThrMultiply;
    const int COUNT_THREAD = 2;
    HANDLE arrayHandleThread[COUNT_THREAD];
    unsigned long uThrPlusID;
    unsigned long uThrMultiplyID;

    hEvent = CreateEvent(NULL, FALSE, TRUE, NULL);      //2-ой параметр -
указывает, что автосброс; 3-ий - что событие свободно
    hThrMultiply = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)ThreadMultiply, expression, 0, &uThrMultiplyID);
    if (hThrMultiply != NULL){
        std::cout << "\n1-ый поток создался\n";
        hThrPlus = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadPlus,
expression, 0, &uThrPlusID);
        if (hThrPlus){
            std::cout << "\n2-ой поток создался\n";
        }
        else{
            std::cout << "\n2-ой поток не удалось создать\n";
```

```
        }
    }
    else{
    std::cout << "\n1-ый поток не удалось создать! Поэтому не будем создавать
2-ой!";
    }
    //заносим в массив дескрипторов, чтобы отследить, когда потоки завершат
свою работу
    arrayHandleThread[0] = hThrMultiply;
    arrayHandleThread[1] = hThrPlus;
    //ждем когда потоки завершат свою работу
    WaitForMultipleObjects(COUNT_THREAD, arrayHandleThread, TRUE, INFINITE);
    std::cout << "\n_____________Вывод_____________\n";
    for (int i = 0; expression[i][0] != '\0'; i++){

        std::cout << expression[i];
    }
    //закрываем потоки
    CloseHandle(hThrMultiply);
    CloseHandle(hThrPlus);
    CloseHandle(hEvent);
    std::cout << "\n";
    system("pause");
    return 0;
}
```

Когда я создаю два потока:

hThrMultiply = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadMultiply, expression, 0, &uThrMultiplyID);

hThrPlus = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadPlus, expression, 0, &uThrPlusID);

Как указать, чтобы **hThrMultiply** запустился ПЕРВЫМ?

Синхронизация процессов при помощи мютексов.

Мютекс представляет собой объект ядра, имеющий свое собственное имя и предназначенный для синхронизации потоков в различных процессах. По технологии работы мьютекс очень похож на критическую секцию:
Вызвать ф-ию для создания мютекса
HANDLE CreateMutex(
LPSECURITY_ATTRIBUTES lpMutexAttributes, // pointer to security attributes
BOOL bInitialOwner, // начальный владелец
LPCTSTR lpName // pointer to mutex-object name);
bInitialOwner == TRUE означает. что объект, который создал мьютекс явл-ся его владельцем; если FALSE, то он свободен и его владельцем будет тот, кто первым его захватит.
lpName – текстовая строка. которая указывает имя мютекса.
В кач-ве ф-ии, описывающей мютекс м/б использована ф-ия OpenMutex(), но чаще исп-ся CreateMutex() – если мьютекс не создан, то он создается.

Для того, чтобы определить возможность входа в критич. уч-к кода, исп-ся одна из ф-ий ожидания; в качестве HANDLE передается HANDLE мьютекса. В случае ожидания ф-ия может вернуть дополнительно одно значение **WAIT_ABANDONED**. Это означает, что мьютекс был освобожден (стал доступен) в рез-те завершения процесса, захватившего мьютекс без вызова ф-ии **ReleaseMutex().** В подавляющем большинстве случаев подобное значение сигнализирует об ошибке.
После использования мьютекса, его нужно освободить вызовом ф-ии:

BOOL ReleaseMutex(HANDLE hMutex); // handle of mutex object

Затем объект синхронизации мьютекс д/б уничтожен вызовом ф-ии;
BOOL CloseHandle(HANDLE hObject); // handle to object to close

Пример: использование мьютекса

```
// Ф-ия потока, которая будет использовать мьютекс
VOID WINAPI F1(…) {
HANDLE h;
h = CreateMutex(NULL,  TRUE, “M1”);
switch(WaitForSingleObject(h,  INFINITE)) {
case WAIT_OBJECT_0 T:
BeginProcess();
break;
case WAIT_ABANDONED:
CloseHandle(h);
return;
}
ReleaseMutex(h);
CloseHandle(h);
}
```