**Министерство цифрового развития, связи и массовых**

**коммуникаций Российской Федерации**

**Ордена Трудового Красного Знамени**

**федеральное государственное бюджетное**

**учреждение высшего образования**

**«Московский Технический Университет Связи и Информатики»**

Кафедра «Системное программирование»

**ЛАБОРАТОРНАЯ РАБОТА №4**

по дисциплине «Операционные системы»

по теме «Методы синхронизации потоков»

Выполнил: студент группы БСТ2001

Савкин Д. И.

Проверил: асс. каф. СП

Алексанян Д. А.

Москва 2022

# Цели и задачи

Целью работы является получение практических навыков использования Win32 API для синхронизации потоков.

Задачей является разработка программного продукта, который демонстрирует применение следующих средств синхронизации: критическая секция, мьютекс, событие и семафор.

# Ход выполнения работы

Исходный код конечного продукта представлен ниже.

#include <stdio.h>

#include <stdint.h>

#include <stdarg.h>

#include <windows.h>

#pragma warning(disable: 4305)

#define fatal\_error(fmt, ...) do { error(fmt, \_\_VA\_ARGS\_\_); exit(1); } while (0)

#define For(x) for (size\_t i = 0; i < (x); ++i)

static void error(const char\* const fmt, ...) {

va\_list args;

va\_start(args, fmt);

printf("ERROR: ");

vprintf(fmt, args);

printf("\n");

va\_end(args);

}

static uint32\_t counter;

static CRITICAL\_SECTION section;

static HANDLE mutex, event, semaphore;

static DWORD WINAPI critical\_section\_thread\_routine(LPVOID param) {

(void)param;

EnterCriticalSection(&section);

counter++;

printf("[critsec] counter is %u\n", counter);

LeaveCriticalSection(&section);

return 0;

}

static DWORD WINAPI mutex\_thread\_routine(LPVOID param) {

(void)param;

WaitForSingleObject(mutex, INFINITE);

counter++;

printf("[mutex] counter is %u\n", counter);

if (!ReleaseMutex(mutex)) {

fatal\_error("unable to release mutex (code %lu)", GetLastError());

}

return 1;

}

static DWORD WINAPI event\_thread\_routine(LPVOID param) {

(void)param;

WaitForSingleObject(mutex, INFINITE);

counter++;

printf("[event] counter is %u\n", counter);

return 1;

}

static DWORD WINAPI semaphore\_thread\_routine(LPVOID param) {

(void)param;

WaitForSingleObject(semaphore, INFINITE);

counter++;

printf("[semaphore] counter is %u\n", counter);

if (!ReleaseSemaphore(semaphore, 1, NULL)) {

fatal\_error("unable to release semaphore (code %lu)", GetLastError());

}

return 1;

}

static void WaitAndReleaseThreads(HANDLE\* threads, size\_t num\_threads) {

WaitForMultipleObjects(num\_threads, threads, TRUE, INFINITE);

For(num\_threads) { CloseHandle(threads[i]); }

}

int main(int argc, char\* argv[]) {

if (argc != 2) {

fatal\_error("Usage: lab04.exe <threads-number>");

}

uint32\_t num\_threads = strtoul(argv[1], NULL, 10);

if (num\_threads == ULONG\_MAX) {

num\_threads = MAXIMUM\_WAIT\_OBJECTS;

} else if (num\_threads > MAXIMUM\_WAIT\_OBJECTS) {

fatal\_error("The number of threads cannot exceed %d", MAXIMUM\_WAIT\_OBJECTS);

}

// NOTE(wkns37): Critical Section

DWORD tid;

HANDLE threads[MAXIMUM\_WAIT\_OBJECTS] = { 0 };

InitializeCriticalSection(&section);

For(num\_threads) {

threads[i] = CreateThread(NULL, 0, critical\_section\_thread\_routine, NULL, 0, &tid);

}

WaitAndReleaseThreads(threads, num\_threads);

DeleteCriticalSection(&section);

// NOTE(wkns37): Mutex

mutex = CreateMutexA(NULL, FALSE, NULL);

if (mutex == NULL) {

fatal\_error("unable to create mutex (code %lu)", GetLastError());

}

For(num\_threads) {

threads[i] = CreateThread(NULL, 0, mutex\_thread\_routine, NULL, 0, &tid);

}

WaitAndReleaseThreads(threads, num\_threads);

// NOTE(wkns37): Event

event = CreateEventA(NULL, FALSE, TRUE, NULL);

if (event == NULL) {

fatal\_error("unable to create event (code %lu)", GetLastError());

}

For(num\_threads) {

threads[i] = CreateThread(NULL, 0, event\_thread\_routine, NULL, 0, &tid);

}

WaitAndReleaseThreads(threads, num\_threads);

// NOTE(wkns37): Semaphore

semaphore = CreateSemaphoreA(NULL, 1, 1, NULL);

if (semaphore == NULL) {

fatal\_error("unable to create semaphore (code %lu)", GetLastError());

}

For(num\_threads) {

threads[i] = CreateThread(NULL, 0, semaphore\_thread\_routine, NULL, 0, &tid);

}

WaitAndReleaseThreads(threads, num\_threads);

return 0;

}

# Заключение

Результат работы программного продукта представлен на рисунках ниже.

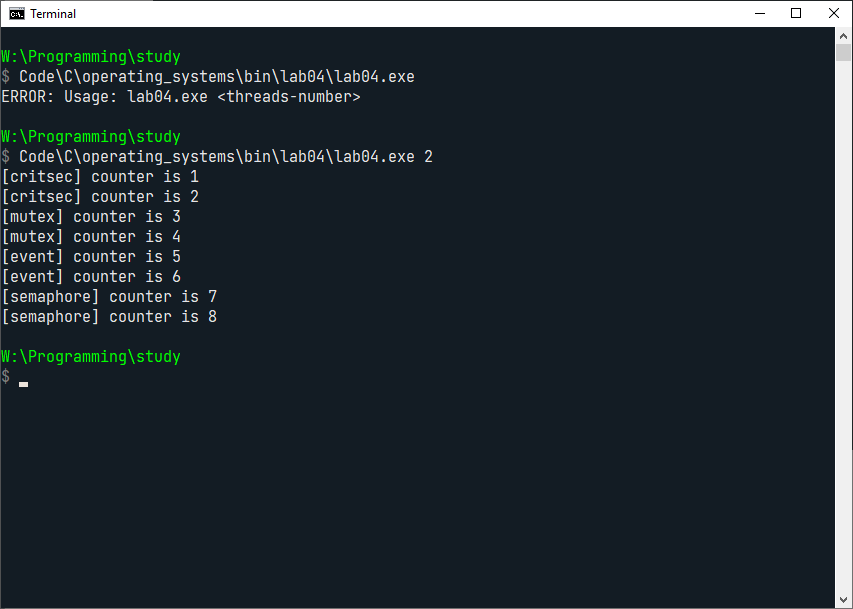


Рисунок 1 – Результат работы программы