**4 ЛАБОРАТОРНАЯ РАБОТА №4**

**«Синхронизация процессов и потоков в ОС Windows.**

**Семафоры.»**

**4.1 Цель работы**

Изучение программного интерфейса приложений (API) операционных систем Windows 9x, NT, ME, XP, 7. Приобретение практических навыков синхронизации потоков, с использованием семафоров в средах программирования Borland Delphi, C++ Builder или Visual Studio.

**4.2 Вариант задания – 17**

Написать программу, содержащую два потока. Первый поток генерирует последовательность чисел и помещает их в кольцевой\* буфер из Nbuf элементов (с проверкой на свободное место в буфере с использованием механизма семафоров). Второй считывает данные из буфера и выводит их на экран. \*) при заполнении кольцевого буфера добавление элементов продолжается сначала, т.е. для вычисления индекса очередного элемента используется операция деления по модулю Nbuf(Pascal: i mod Nbuf; Cи: I % Nbuf).

Количество элементов (N) – 600, тип последовательности – Арифметическая прогрессия с разностью 1 (Натуральные числа), длина буфера (N\_buf) – 9.

**4.3 Ход работы**

4.3.1 В соответствии с вариантом была написана программа на языке си и её текст представлен ниже.

#include <stdio.h>

#include <windows.h>

#include <time.h>

#define N\_buf 9

#define N 600

#define USE\_SEMAPHORE

int arr[N\_buf];

DWORD WINAPI Generate\_numbers\_and\_put\_to\_buffer (LPVOID p) {

#ifdef USE\_SEMAPHORE

HANDLE sem;

sem = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, "My\_Semen");

#endif

srand(time(NULL));

int random\_num = rand()%399;

for (int i = 0; i < N; i++) {

#ifdef USE\_SEMAPHORE

WaitForSingleObject(sem, INFINITE);

#endif

arr[i % N\_buf] = random\_num + i;

}

}

DWORD WINAPI Read\_data\_and\_show\_on\_the\_screen (LPVOID p) {

#ifdef USE\_SEMAPHORE

HANDLE sem;

sem = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, "My\_Semen");

#endif

for (int i = 0; i < N; i++) {

printf ("%d ", arr[i % N\_buf]);

#ifdef USE\_SEMAPHORE

ReleaseSemaphore(sem, 1, NULL);

#endif

if ((i % N\_buf) == 8) {

printf ("\n");

Sleep(40);

}

}

printf ("\n");

}

int main() {

HANDLE th[2], sem;

#ifdef USE\_SEMAPHORE

sem = CreateSemaphore(NULL, N\_buf, N\_buf, "My\_Semen");

#endif

th[0] = CreateThread(NULL, 0, Generate\_numbers\_and\_put\_to\_buffer, NULL, 0, NULL);

th[1] = CreateThread(NULL, 0, Read\_data\_and\_show\_on\_the\_screen, NULL, 0, NULL);

WaitForMultipleObjects(2, th, TRUE, INFINITE);

#ifdef USE\_SEMAPHORE

CloseHandle(sem);

#endif

system("pause");

return 0;

}

4.3.2 Тестовые примеры продемонстрированы на рисунках 4.1 – 4.2.

Сперва была написана программа без использования семафоров. Ожидаемо результат выполнения программы оказался не удовлетворительным, по причине рассинхронизации потоков (Рис. 4.1).

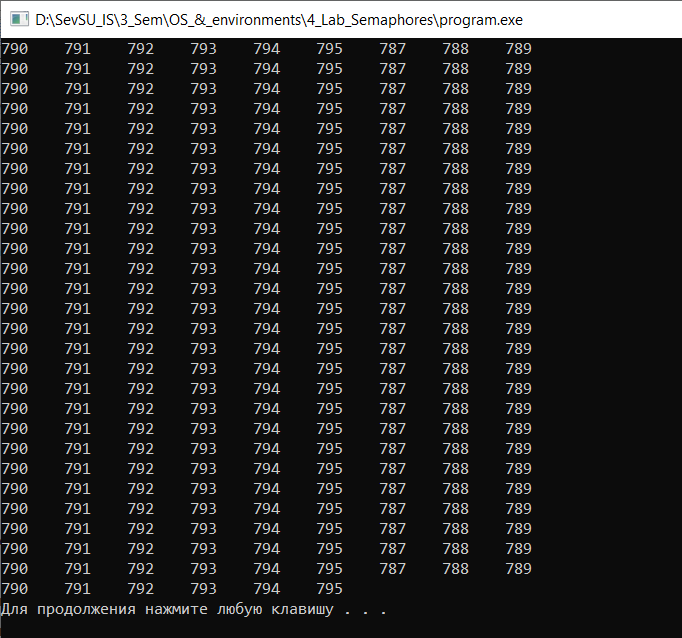


Рисунок 4.1 ­– Результат выполнения программы без применения семафоров

После чего в программу были добавлены семафоры. После перекомпиляции программы с использованием семафоров на экран были выведены корректные данные, что продемонстрировано на рисунке 4.2.

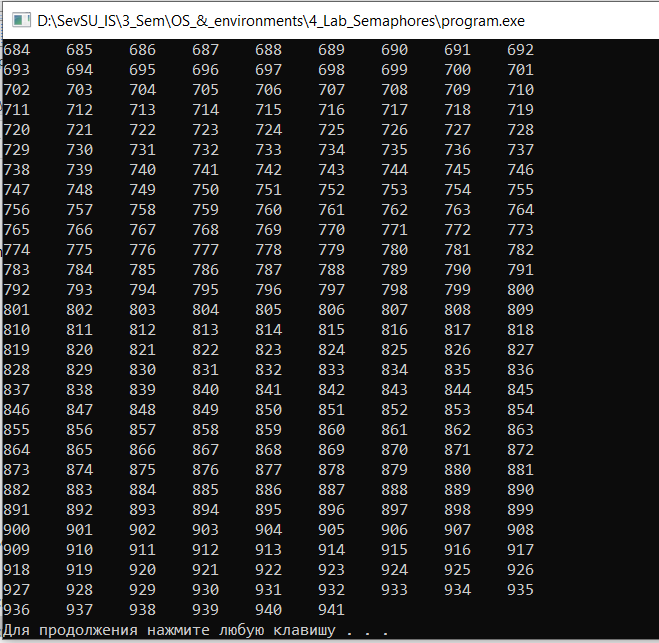


Рисунок 4.2 ­– Результат выполнения программы с применением семафоров

Результаты тестирования полностью соответствуют ожиданиям.

**Выводы**

При выполнении данной лабораторной работы были получены навыки разработки программ с семафорами с помощью средств WinAPI. Определено, что в WinAPI аналогом операции P(S) являются WaitForSingleObject и WaitForMultipleObjects, а операции V(S) – ReleaseSemaphore. Получены практические навыки работы с семафорами. Также были повторно закреплены навыки использования многопоточности. Полученные знания позволят создавать многопоточные приложения, которые на многоядерных системах будут работать эффективнее.