# Лабораторная работа №4

Синхронизация процессов и потоков в ОС Windows. Семафоры

## 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.

## Постановка задачи

Приобрести практические навыки синхронизации потоков с помощью семафоров.

## Текст задания

(Вариант 19) 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) – 240 –, тип последовательности – Арифметическая прогрессия с разностью 3 –, длина буфера (N\_buf) – 5.

## Ход выполнения работы

### Тексты программ

В соответствии с выбранным вариантом программа была написана в среде программирования C++ Builder. Текст программы представлен ниже, в Листинге 1.

Листинг 1 для программы Master4:

/\*

USE\_SEMAPHORE -> При компиляции использовать семафоры

RANDOM\_PROGRESSION -> Установить случайным первый член прогрессии

WRITE\_DELAY -> Увеличить время записи в буфер

PRINT\_DELAY -> Увеличить время печати из буфера

\*/

#include <iostream>

#include <ctime>

#include <windows.h>

#define N\_buf 5

#define N 240

#define PROGRES\_DIFF 3

#define NSEM\_FREE "MySem\_Free"

#define NSEM\_FULL "MySem\_Full"

#define NSEM\_BOOL "MySem\_Bool"

using std::cout;

using std::endl;

typedef int elem\_t;

elem\_t arr[N\_buf];

DWORD WINAPI print\_from\_buff (LPVOID);

DWORD WINAPI write\_to\_buff (LPVOID);

int main ()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

HANDLE hth[2], sem[3];

#ifdef USE\_SEMAPHORE

sem[0] = CreateSemaphore(NULL, N\_buf, N\_buf, NSEM\_FREE);

sem[1] = CreateSemaphore(NULL, 0, N\_buf, NSEM\_FULL);

sem[2] = CreateSemaphore(NULL, 1, 1, NSEM\_BOOL);

#endif

// Создать 2 потока

hth[0] = CreateThread(NULL, 0, print\_from\_buff, NULL, 0, NULL);

hth[1] = CreateThread(NULL, 0, write\_to\_buff, NULL, 0, NULL);

// Ждать завершения работы этих потоков

WaitForMultipleObjects(2, hth, TRUE, INFINITE);

for (int i = 0; i < 3; i++)

CloseHandle(sem[i]);

return 0;

}

DWORD WINAPI print\_from\_buff (LPVOID p)

{

#ifdef USE\_SEMAPHORE

HANDLE sem[3];

sem[0] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_FREE);

sem[1] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_FULL);

sem[2] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_BOOL);

#endif

for (size\_t i = 0; i < N; i++)

{

#ifdef USE\_SEMAPHORE

// Запрос на увеличение заполненности буфера

WaitForSingleObject(sem[1], INFINITE);

// Запрос на переход в монопольный режим работы над буфером

WaitForSingleObject(sem[2], INFINITE);

#endif

cout << arr[i % N\_buf] << " ";

#ifdef WRITE\_DELAY

// Искусственная задержка записи

Sleep(100);

#endif

#ifdef USE\_SEMAPHORE

// Уменьшить свободность буфера

ReleaseSemaphore(sem[0], 1, NULL);

// Выйти из монопольного режима

ReleaseSemaphore(sem[2], 1, NULL);

#endif

}

cout << endl;

return 0;

}

DWORD WINAPI write\_to\_buff (LPVOID p)

{

#ifdef RANDOM\_PROGRESSION

srand(time(NULL));

#else

srand(1);

#endif

#ifdef USE\_SEMAPHORE

HANDLE sem[3];

sem[0] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_FREE);

sem[1] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_FULL);

sem[2] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, NSEM\_BOOL);

#endif

elem\_t curr = rand() % 5 - PROGRES\_DIFF;

for (size\_t i = 0; i < N; i++)

{

#ifdef USE\_SEMAPHORE

// Запрос на увеличение свободности буфера

WaitForSingleObject(sem[0], INFINITE);

// Запрос на переход в монопольный режим работы над буфером

WaitForSingleObject(sem[2], INFINITE);

#endif

arr[i % N\_buf] = (curr += PROGRES\_DIFF);

#ifdef PRINT\_DELAY

// Искусственная задержка вывода

Sleep(250);

#endif

#ifdef USE\_SEMAPHORE

// Уменьшить заполненность буфера

ReleaseSemaphore(sem[1], 1, NULL);

// Выйти из монопольного режима

ReleaseSemaphore(sem[2], 1, NULL);

#endif

}

return 0;

}

### Тестовые примеры

Вначале программа не поддерживала работу с семафорами. Рисунок 4.1 содержит вывод программы. Был сделан вывод, что функция заполнения буфера работает слишком быстро и поэтому до печати элементов прогрессии на экран функция успевает перебрать все члены прогрессии и оставить в буфере лишь «706 709 712 715 718». Функции печати остаётся лишь несколько раз вывести эти пять чисел.

После этого программа была переписана для поддержки работы с семафорами (компиляция с флагом -DUSE\_SEMAPHORE). Рисунок 4.2 показывает вывод новой версии программы. Вывод арифметической прогрессии верен.

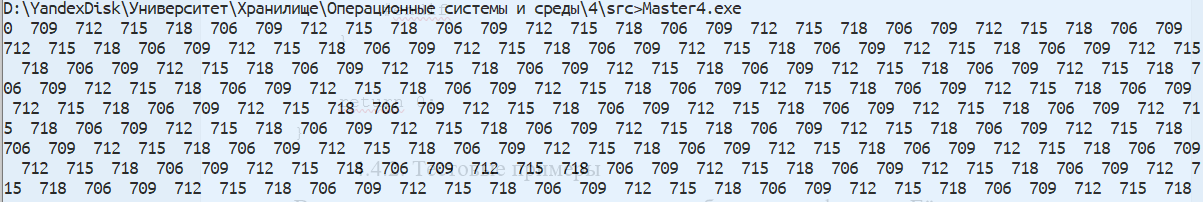


Рисунок 4.1 – Вывод программы (семафоры не используются)

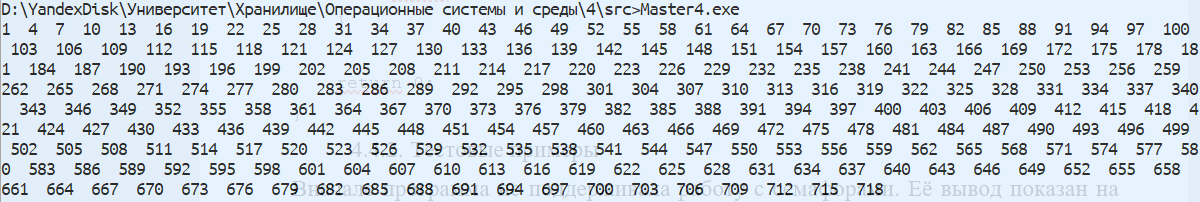


Рисунок 4.2 – Вывод программы (с использованием семафоров)

Были проведены тесты, в которых вывод из буфера и запись в буфер были искусственно замедлены с помощью Sleep() (поочерёдная компиляция с флагами -DPRINT\_DELAY и -DWRITE\_DELAY). Рисунок 4.3 и Рисунок 4.4 содержат выводы этих программ соответственно. Было сделано заключение, что вывод программы не зависит от времени, затраченного на запись в буфер и печать из него.

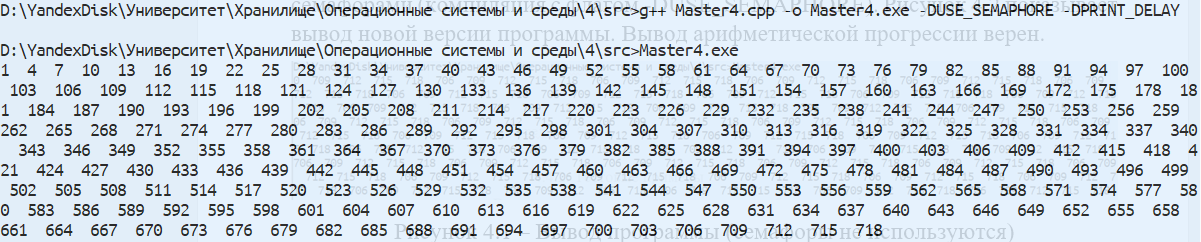


Рисунок 4.3 – Вывод программы с искусственной задержкой вывода из буфера

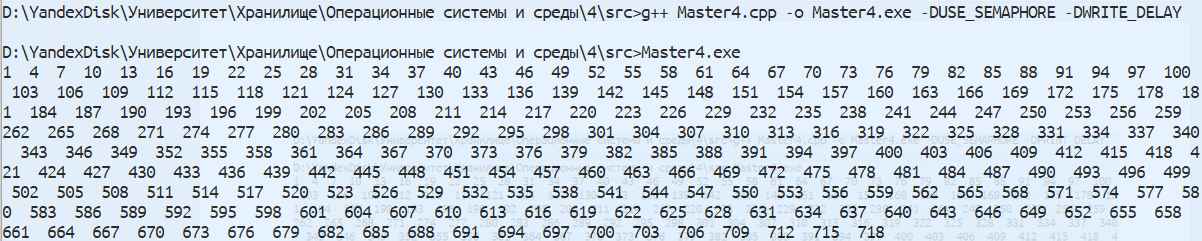


Рисунок 4.4 – Вывод программы с искусственной задержкой записи в буфер

## Вывод

При выполнении данной лабораторной работы были получены навыки работы 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.